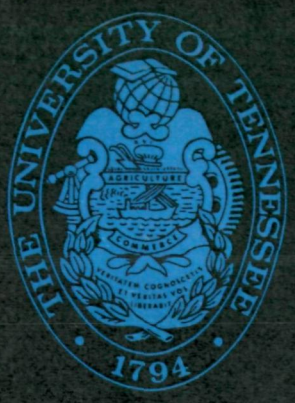


ELECTRICAL ENGINEERING DEPARTMENT

PROGRAMMABLE DATA COLLECTION PLATFORM STUDY

Final Report

May 1976

UNIVERSITY OF TENNESSEE

KNOXVILLE
TN 37916

The University of Tennessee
Department of Electrical Engineering
Knoxville, Tennessee 37916

PROGRAMMABLE DATA COLLECTION PLATFORM STUDY

Final Report

May 1976

Contract No. NAS5-22495

Prepared for

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

Page
Intentionally
Left Blank

PREFACE

This report represents the findings of the work completed under a nine-month study of programmable data collection platforms. The system has been implemented with a microcomputer, and the results show that programmable data collection platforms can carry out all the functions of hardwired data collection platforms with capacity left to perform a number of desirable computational functions.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

ILLUSTRATIONS

ILLUSTRATIONS (Continued)

PROGRAMS

TABLES

1. INTRODUCTION

1.1 BACKGROUND AND PURPOSE

Data collection by satellite is a rather young branch of science which was first demonstrated in 1967 using the ATS-1 satellite. This first NASA demonstration was the Omega Position Location Equipment System (OPLE) which primarily determined that an accurate position fix could be obtained from platforms in remote locations. This system was followed by the Interrogation, Recording and Location System (IRLS) flown on the Nimbus-3 satellite in 1969. The IRLS was closely followed by the French Eole satellite which was solely devoted to the tasks of data collection and position location for remote platforms distributed around the globe.

One fact that stood out from these early experiments was that the user costs for platform and data reduction must be kept low if the satellite data collection concepts are to be utilized on a large-scale basis. A way to reduce costs was incorporated by the Landsat data collection system flown in 1972. The platform transmissions were random rather than ordered. This simplified the platforms by eliminating on-board receivers and reduced the costs substantially. In 1974 the GOES satellite carried a data collection system which again utilized an ordered system, but costs were kept low because of improvements in semiconductor technology.

In 1975 the Nimbus-6 satellite carried the Tropical Wind Energy-conversion and Reference Level Experiment (TWERLE) into orbit. The TWERLE utilized the low cost of the random transmission system combined with new low-cost integrated circuits to maintain a still lower platform cost.

In 1977 the TIROS-N satellite will be launched carrying a data collection system designed by France. The system will use the random transmission feature which will help to lower user costs and aid in making the system internationally acceptable.

A primary purpose of this study is to determine if the recent advances in semiconductor technology can be incorporated in the design of data collection platforms to further reduce their cost and, consequently, make their application more desirable to the user.

At the time of the instigation of this study, all data collection platforms used hardwired logic circuitry to implement the identification and formatting of the data collected by a platform. Hardwired circuitry was also used to control the sensors and the transmitter. In the last two years, a new semiconductor device which has the potential of drastically reducing hardware costs while increasing the capability of the data collection platform has appeared on the commercial market. The device is the microprocessor. The microprocessor is a computer central-processing unit (CPU) on a single integrated circuit chip. The equivalent of thousands of discrete electronic components are fabricated on a single microprocessor chip with unit costs projected to be in the ten-dollar range.

The goals of this study are to determine what present data collection functions can be accomplished by substituting a microprocessor for most of the hardwired logic, to uncover new tasks which would enhance the utility of data collection platforms by virtue of having a microprocessor available, and to determine if the programmable feature of the microprocessor will allow future platforms to be compatible with more than one satellite data collection system. To prove the concepts developed in achieving these goals, a model programmable data collection platform (PDCP) development system has been implemented. The details of the University of Tennessee (UT) PDCP development system are described in Section 5 of this report.

1.2 SUMMARY

This report describes the results of a study of the feasibility of incorporating microprocessors in data collection platforms (DCP's). An introduction to microcomputer hardware and software concepts is provided in Section 2. Thus, readers who are not familiar with the microprocessor

field are furnished necessary background information on the basic organization of a microcomputer and software development techniques.

Programmable data collection platform (PDCP) hardware design goals include minimizing power consumption, maintenance, weight, and cost while providing accurate and reliable operation. Section 3 discusses the influence of microprocessor technology on the design of PDCP hardware. A standard, modular PDCP design capable of meeting the design goals listed above is proposed. The microprocessor contributes to this design by minimizing PDCP control logic and simplifying sensor interfaces. Also, standard PDCP sensor and transmitter interfaces will promote further cost reductions.

Although the standard, modular PDCP design proposed in Section 3 is economically desirable, the PDCP must be sufficiently flexible to operate efficiently with a number of different sensors, data compaction techniques, and data transmission formats. The PDCP software described in Section 4 is the key to attaining these goals. Numerous examples of potential PDCP programs are presented to demonstrate that a microprocessor is capable of performing all tasks of current DCP random logic control units. Furthermore, the PDCP software system will contribute to reduced hardware costs by replacing random logic and complex sensor interfaces with inexpensive program memory. In addition, a PDCP can economically perform data compaction operations that are impractical for random logic implementation. This will allow users to obtain more information from each PDCP without exceeding transmission channel bandwidth limitations.

Section 4 also discusses the process of developing PDCP programs. A specific PDCP software organization designed to minimize software development costs is described. Traditional program development techniques are inadequate for PDCP software development by applications oriented users. A PDCP software development system which would allow applications oriented users to define the software structure of their individual PDCP's in familiar terms is recommended for a future study. The proposed editor/translator software development system retains the

efficiency of assembly language programming because users are allowed to create software systems from a library of subroutines written by experienced assembly language programmers.

An integral part of this study was the design and construction of the UT PDCP development system described in Section 5. This system is useful in the development, evaluation, and demonstration of potential PDCP programs. The UT PDCP provides all the capabilities of a PDCP control unit. Programs developed for use on the UT PDCP include sensor data input subroutines, data compaction subroutines, and data transmission subroutines. The UT PDCP is intended for use primarily as a program development and demonstration system. Therefore, the design of the system is not intended to represent the design of an actual PDCP. The system will serve as a machine through which NASA personnel can acquaint themselves with the details of PDCP software and PDCP software development.

A PDCP design should be based on a knowledge of currently available microprocessors and projected future microprocessors. Section 6 describes a weighting matrix technique for evaluating microprocessors and provides a microprocessor technology forecast covering the next five years. The weighting matrix provides a systematic procedure for generating PDCP application performance measures for the various microprocessors.

2. MICROCOMPUTER HARDWARE AND SOFTWARE

The control logic found in most current data collection platform designs is an example of the custom random logic approach to process control system design. Custom random logic combines individual logic elements (such as flip-flops, gates, counters, etc.) to perform a particular system control task. The primary tasks of the random logic of a DCP are to control the sensors, input sensor data, format the data, and control the DCP transmitter. CMOS logic is prevalent in current DCP designs due to the extremely low power requirements for CMOS circuits.

A major disadvantage of current hardwired DCP's is that they are based on customized designs which are intended to perform only specific tasks. In practice, applications for DCP's vary considerably due to the diversified requirements of the various users. As a result, the introduction of new DCP applications or a new data collection satellite can require the development of a new DCP design. This increases DCP costs due to both the increase in direct development cost and the increase in manufacturing cost resulting from the production of smaller quantities of each specific design.

What are the alternatives for a state-of-the-art, cost-effective DCP design? A close look at modern process control systems suggests an answer. Until the introduction of the microprocessor, state-of-the-art process control systems depended on custom built random logic designs or the dedicated minicomputer.

Custom built random logic can be cost effective if a large number of identical systems are required where the initial high design costs can be effectively shared by a large number of users. This approach certainly has merit in the DCP field, but the question still remains as to whether a more versitile and perhaps more cost effective approach exists with modern technology. Also, a DCP user requiring only a small number of platforms must pay an extremely high price for a new design unless a present design can satisfy his requirements.

The dedicated minicomputer approach would provide versitility in a DCP design as modification of the DCP tasks could be implemented in software. Unfortunately, the minicomputer is physically too large and heavy to be practical in a DCP design. Also, power requirements for the minicomputer cannot be met by the typical DCP power supply.

The microprocessor is causing a revolution in the process control field [1-4].* A microprocessorized DCP would be programmable like the minicomputer, yet cost, size, weight, and power requirements would be greatly reduced. Produced in quantity, a microprocessorized process control system provides reduced hardware costs over an equivalent custom random-logic process-control system. This is due mainly to a reduction in the number of components required [1-3, 5-7].

A microprocessor-based DCP could not only provide reduced hardware costs, but later modification of the DCP task could be achieved at a significantly lower cost compared to a custom random-logic design. The cost of redesigning a custom random logic system could well exceed the initial design cost. The microprocessor-based DCP would probably require only a software revision. Hardware cost would be limited to replacement or reprogramming of one or more read-only memories (ROM's). Software costs should in general be low compared to high redesign and hardware costs for a custom random logic-system.

Besides providing great flexibility [3, 4] in altering the task performed by the DCP, a microprocessor-based design offers the added advantage of providing data processing capability at a small additional cost. Providing data processing capability would require adding the necessary software to perform the additional task. Again, hardware cost would be minimal requiring only the addition of one or more ROM's. Software cost could be spread out among the large number of DCP users. Field programmable ROM's would be used in prototype models, and cost effective mask-programmed ROM's would be used in production DCP's to minimize hardware costs. Further hardware aspects of the proposed programmable DCP (PDCP) are discussed later in this report (Section 3.2).

*References are located at the end of each chapter.

To fully appreciate the potential capabilities and advantages of a microprocessor based DCP, one must be familiar with the basic concepts of a general computer system and the specific hardware and software characteristics of a microprocessor. Therefore, the remainder of this section is devoted to a discussion of the basic characteristics of a microcomputer.

The definition of a microcomputer differs from author to author; however, a suitable definition for the purpose of this study is that a microcomputer is a computer whose major component, the central processing unit, is a single microprocessor chip or microprocessor chip set. To complete the definition, the term computer must be defined.

Briefly, a computer is a device or machine which performs a programmed sequence of operations on data. A computer is comprised of three major components:

- Central Processing Unit (CPU)
- Memory
- Input/Output (I/O)

Instructions and data are placed in the computer memory. The central processing unit (CPU) fetches instructions from the memory and executes the instructions. The instructions are programmed to cause the CPU to process data. The set of instructions executed by the CPU is called a program. Data may be initially in memory, processed by the CPU, and results returned to memory. Input/output ports of the computer provide a means of entering and retrieving the data from the CPU and/or the memory. Note that data can be control words or signals as well as numerical quantities. Thus, the computer is capable of performing the following tasks:

1. Input data and control signals
2. Process and format data
3. Output control signals and data

In particular, a computer could input DCP sensor data; process the data (convolutional encoding, Manchester encoding, data compaction, etc.); provide and recognize DCP control signals; and format the platform data to emulate any of the typical DCP data formats.

Sections 2.1 and 2.2 discuss the hardware architecture and software aspects of the typical microprocessor. These sections reveal the processing and control capabilities of the microcomputer and provide insight into the question as to whether the microcomputer is indeed capable of performing the task of a typical DCP.

2.1 HARDWARE ARCHITECTURE OF THE MICROPROCESSOR

There are three basic components of the typical central processing unit (CPU) as depicted in Figure 2.1(1).

- Arithmetic/Logic Unit (ALU)
- Registers (Temporary Storage)
- Control Logic

The arithmetic/logic unit (ALU), as the name implies, performs the arithmetic and logical operations on the data processed by the CPU. Registers provide temporary internal storage for operands and results as well as address pointers for input/output and memory. The control logic provides the various signals for initiating processor functions and controlling external circuits. Recognition of external control signals is also a function of the control logic. The next three subsections explain in detail the functions of the ALU, registers, and control logic.

2.1.1 Arithmetic/Logic Unit (ALU)

All microprocessors have an arithmetic/logic unit (ALU) which performs the arithmetic and logical operations. As shown in Figure 2.1.1(1), the ALU generally has two multibit inputs.

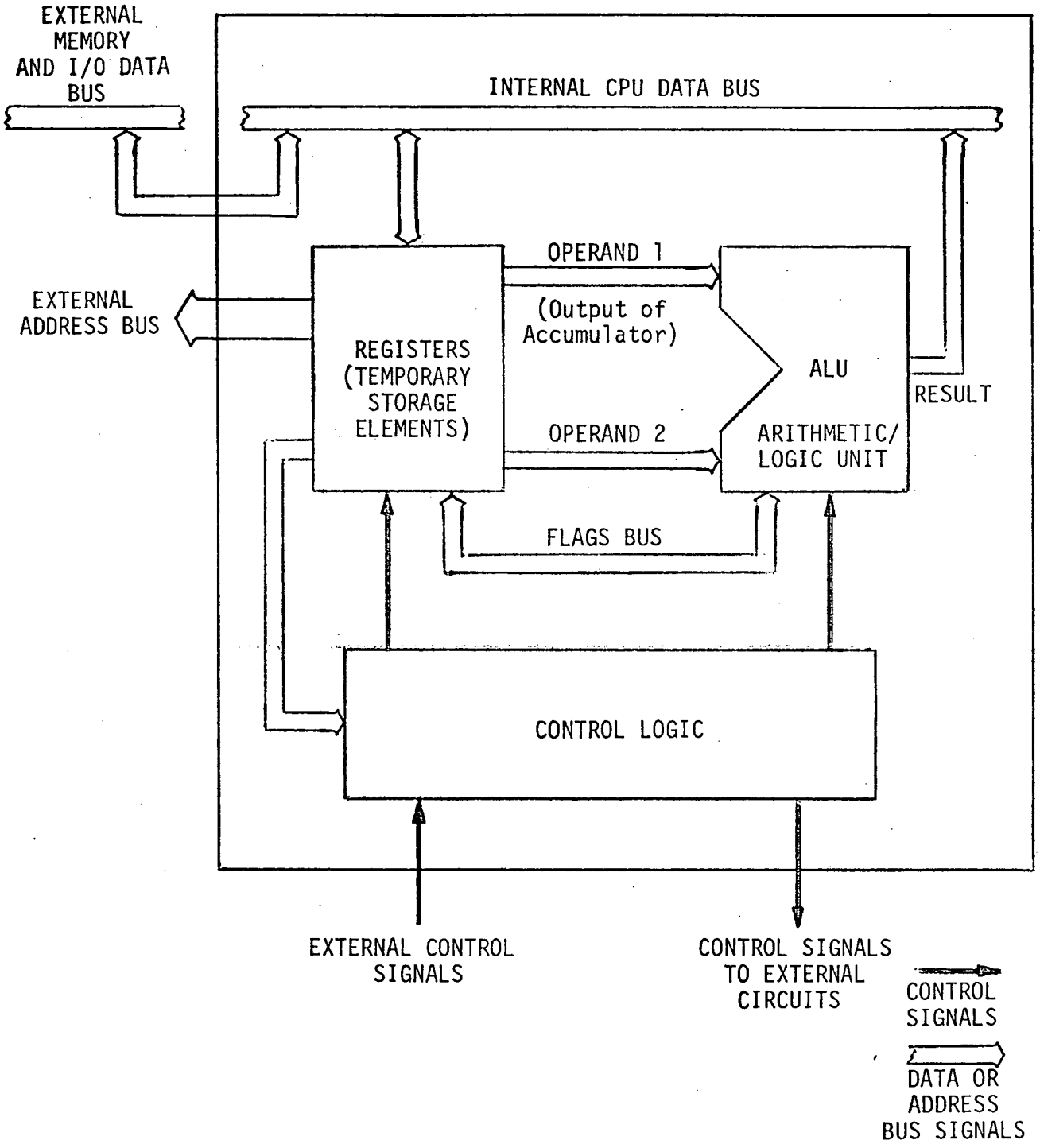


Figure 2.1(1) General Block Diagram of CPU.

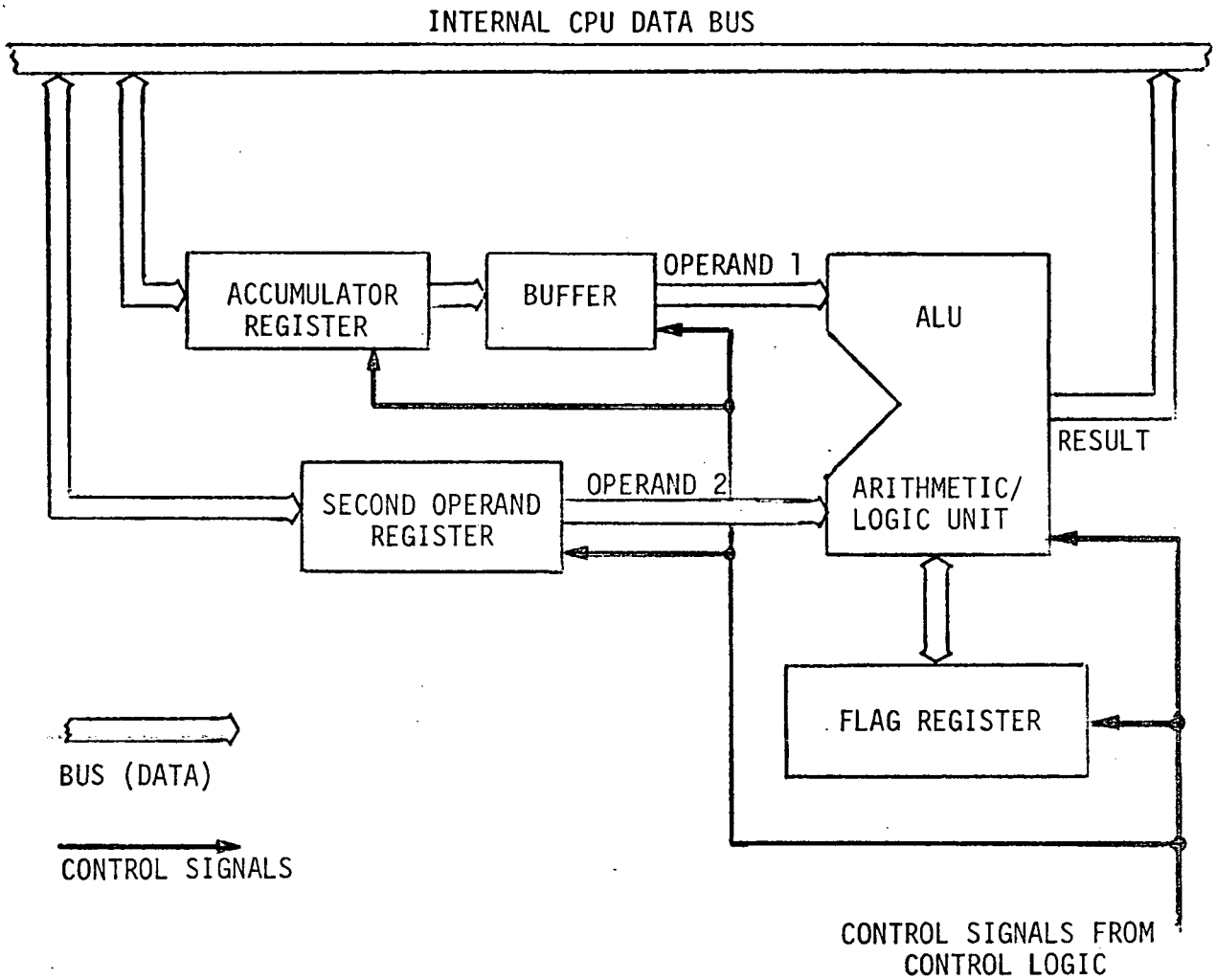


Figure 2.1.1(1) Typical Arithmetic/Logic Unit.

One input (Operand No. 1) to the ALU is usually the output of an accumulator buffer. The buffer stores the accumulator data throughout the ALU operation. Typically, all arithmetic operations are performed between the accumulator register and a temporary register, which in some CPU's may be a second accumulator. Input to the accumulator or second operand register may be memory data, an input port, or some other hardware register. The result of the operation is placed on the internal CPU data bus. Often the result is returned to the accumulator. In this case, the accumulator requires a buffer so that the original data in the accumulator can be stored and applied to the ALU while the arithmetic or logical operation is being performed. Thus, the result can be returned to the accumulator even though the result is a function of the accumulator's original contents.

All ALU's perform the basic arithmetic function of binary addition. Additional arithmetic/logical operations performed by most CPU's include binary subtraction, logical AND, logical OR, logical exclusive OR, complement and register bit shift. Although not yet common functions, some of the more recent microprocessors such as Texas Instruments' TMS 9900 also include binary multiplication and division.

A very simple ALU may only provide binary addition. In this case, binary subtraction, multiplication, and division may be performed only after the programmer has written software routines to perform these functions. Routines can be written to perform binary subtraction, multiplication, and division with only the basic binary addition instruction in conjunction with the ALU flags. Algorithms for performing various binary operations in terms of simpler binary operations can be written as needed for a given processing task. For example, multiplication by two is equivalent to a single bit shift left, and division by two is equivalent to a single bit shift right.

Although even the simplest ALU provides binary addition, the number of additional binary operations provided varies widely from microprocessor to microprocessor. One of the fundamental problems in selecting a microprocessor for a particular task is deciding how much power in performing arithmetic/logical operations is sufficient.

Referring again to Figure 2.1.1(1), notice that the data path between the flag register is bidirectional. The operations performed by the ALU produce an output which may be zero, have a carry or borrow, have even or odd parity, or have positive or negative sign. These auxiliary outputs are called flags and are stored in the flag register for subsequent ALU operations. The flag register is thus a source and destination register.

Besides providing additional data to the ALU, the flag register often provides information for the control logic. Program branching may be conditional on the result of an ALU operation (value of a flag). For example, the programmer may wish to call an overflow subroutine if an ALU operation produces a carry. By placing a call on carry instruction after the ALU operation instruction, the overflow subroutine would be called only if the result of the operation produced a carry. References 10-14 discuss the use of flags and conditional branching in more detail.

2.1.2 Registers

The hardware registers internal to the CPU provide temporary storage locations for data, instructions, and address pointers. The number of bits per register is usually an integer multiple of the characteristic word length of the CPU. All computers have a characteristic word length. The characteristic word length is generally determined by the size of the internal storage elements (registers) and the interconnecting buses. As an example, a CPU that has mostly eight-bit registers and eight-bit buses that transfer information between the registers is said to have an eight-bit word length. Some confusion results when the address bus is other than the characteristic word length.

Perhaps a better indication of the word length of the CPU is the maximum number of data bits stored at a single memory address. Most hardware registers of the CPU are usually equal in size to the memory word length. An example is the 8080A used in the UT PDCP development system. The word length of memory is eight bits. The internal general purpose registers in the 8080A CPU are eight bits. Certain address registers such as the program counter and stack pointer are

comprised of 2 eight-bit registers. Registers, then, are simply temporary storage elements in the CPU. There are many types of special purpose registers which are summarized below.

2.1.2.1 Program Counter Register - All CPU's contain a program counter register. This register stores the memory address from which instructions are fetched from memory. Each time an instruction is fetched, the program counter is normally incremented by one to point to the memory location where the next instruction is located. Multibyte instructions may require multiple incrementing of the program counter in order to read the entire instruction (usually accomplished automatically). Some instructions contain immediate data stored sequentially following the instruction. In this case, the program counter is incremented one or more times so that the entire instruction is read in. After the last byte of the instruction is read, the program counter register is incremented again to point to the first byte of the next instruction.

Most CPU's provide at least two means to alter an otherwise sequential fetching of instructions. The jump-type instruction alters the program counter register contents according to a particular addressing mode specified by the jump instruction (see References 10-14). The subroutine call instruction allows program flow to another area of memory, and upon completion of the "called subroutine" the program returns to the instruction following the call. Execution of a subroutine call requires a stack to store the address of the instruction following the call instruction. When a call instruction is encountered, the address of the instruction following the call instruction is stored in the stack. Then the program counter is loaded with the address of the subroutine and the subroutine is executed. Upon completion of the subroutine, the address previously stored in the stack is returned from the stack and placed into the program counter. This action is initiated by placing a return from subroutine instruction as the last executed instruction at the end of the subroutine. This causes program execution to return to the calling program.

2.1.2.2 Stack and Stack Pointer - A CPU that provides subroutine calls must have a stack. A stack is a last in, first out (LIFO) buffer storage element; that is, the last value stored is the first to be read out.

Two types of stacks are found in microprocessors. The hardware stack (e.g., the Intel 8008) is a LIFO buffer storage element internal to the microprocessor integrated circuit. The software stack (e.g., Intel 8080A or Motorola 6800) is a LIFO buffer storage element implemented in the computer random-access memory or some memory external to the CPU integrated circuit.

An advantage of the software stack is that stack size is limited only by the amount of external memory the programmer dedicates to the stack function. In the case of a software stack, an internal (to the CPU) address register called a stack pointer register is used to address the memory area which is software programmable to serve as the CPU stack. The stack pointer register is generally the same size as the program counter register.

Besides storing addresses for subroutine calls, the stack may also serve as a data storage element. For example, in the 8080A, there are "push" and "pop" instructions which provide for storage of the contents of the various hardware registers. An instruction set possessing this capability has a significant advantage over the ones that do not. Suppose an external interrupt request occurs while the CPU is executing a main program. Further, suppose the interrupt service routine may change CPU register contents depending on which device issues the interrupt. Without stack storage of processor status (contents of CPU registers), the interrupt may have to be delayed until a point is reached in the main program where loss of CPU status is not critical. With stack storage of CPU status, the interrupt can be processed almost immediately with processor status "pushed" onto the stack at the beginning of the interrupt service subroutine and CPU status restored by "popping" the stack at the end of the interrupt subroutine.

2.1.2.3 Accumulator and General Purpose Registers - One of the operands used in ALU operations is generally the current contents of the accumulator. The number of bits in the accumulator register is a good indication of the characteristic word length of the CPU.

During an ALU operation, the accumulator register's contents and some other register's contents are applied to the inputs of the ALU [see Figure 2.1.1(1)]. The result of the operation is usually returned to the accumulator via the internal data bus of the CPU.

In contrast to other operand registers which may be input to the ALU, the accumulator contents are, in general, altered following the completion of an ALU operation. The accumulator is a source for operands and, in most cases, also a destination for results. The buffer register following the accumulator provides the temporary storage of the accumulator register data while the ALU operation takes place. This allows the result of the ALU operation to be returned to the accumulator.

Other hardware registers which serve only as a source or destination register (but not both simultaneously) are called general purpose registers. Some microprocessor architectures do not provide general purpose registers, thus requiring all ALU or other CPU operations to be performed between accumulators, memory, and I/O only. Microprocessors lacking general purpose registers often provide more flexible memory addressing modes to allow single or double byte instructions to execute ALU or other operations with different memory locations. In other words, external memory is used for general purpose registers.

2.1.2.4 Instruction Register - The instruction register is used for temporary storage of the instruction fetched from memory. At the beginning of the instruction cycle (see Section 2.1.3), the instruction is fetched from memory and loaded into the instruction register. The word length of the instruction register is thus equal to the word length of memory.

Corresponding to each particular operation the CPU must perform is a unique code called the instruction or operation code. For an n-bit

machine, there are 2^n unique instruction codes that could be used. Thus, an eight-bit machine allows for up to 2^8 (256) unique instruction codes. The codes are stored sequentially in memory and are fetched one by one into the instruction register where they are stored during decoding and execution of the instruction.

2.1.2.5 Address Registers - The stack pointer and program counter are examples of special function address registers. Some CPU's have other address registers which provide absolute memory pointers, relative memory pointers, input port address pointers, and output port address pointers.

In some CPU's, a general purpose address register may be used for many functions. For example, the 8080A has two general purpose registers (H and L) which are absolute memory address pointers; yet, they may also be used for data storage, stack pointer storage, and even double precision shift left and add.

Memory reference instructions require an address register to hold the address of the memory location to be referenced by the instruction. Sometimes this address register is loaded with an address fetched by memory as part of a multibyte instruction. Some CPU's such as the Motorola 6800 provide an address index register which is used as an indexed memory pointer. Data can be stored or retrieved from the address specified by the index register plus or minus a fixed amount (relative addressing).

2.1.3 Control Logic

The primary and most complex component of the CPU is the control logic. The control logic provides the sequential signals that perform the various processing tasks.

A necessary input signal to the control logic of all CPU's is a master clock. Some microprocessors such as the MOS Technology MCS 6502 have an internal clock while others require an external clock. The popular Intel 8080A used in the UT PDCP requires an external two-phase clock.

Other inputs to the control logic include decoded instructions from the instruction register, flags generated by previous ALU operations, and external control signals such as interrupt, wait, or DMA (direct memory access) requests. The function of the control logic is to take all these input signals and output the necessary signals in the proper sequence so as to execute the appropriate processing task.

The CPU operates in a cycle. That is, the processor fetches an instruction, decodes the instruction, executes the instruction, fetches the next instruction, etc. The master clock provides a reference timing signal to synchronize the events in a processor cycle.

An instruction fetch, instruction decode, and instruction execution is often called an instruction cycle. The instruction cycle generally requires one or more subcycles called machine cycles. Each distinct function performed during a machine cycle is called a state. A state generally requires one or more periods of the master clock. The first state of every instruction cycle is an instruction fetch. During the instruction fetch state, the program counter provides the address of the next instruction, and a memory read subcycle places the instruction fetched from memory on the internal data bus. From the data bus, the instruction is loaded into the instruction register. The instruction remains in the instruction register throughout the instruction cycle.

Following the decoding of the instruction, cycle status information is provided by the control logic. Cycle status signals indicate the type of machine cycle that is to be performed. For example, the instruction cycle may be input/output (I/O), memory referencing (memory read or write), internal CPU processing, or a combination of these.

Input/output cycles may not be provided by some microprocessors. This class of microprocessors relies on a technique called memory-mapped I/O. That is, input and output ports are treated like any other memory location. Data sent to or from a memory-mapped I/O port appears to the CPU to have been written into or read from memory. All microprocessors have the capability of memory-mapped I/O. Those microprocessors providing I/O cycles offer a second method to input and output data to and from

the CPU. The I/O cycle is flagged by I/O status bits following the decoding of an I/O instruction. The I/O flag bits combined with the I/O port address provide the signals to select the external I/O ports and whether the operation is to be an input or output. Memory mapped I/O uses memory read or write flag bits combined with reserved memory addresses to operate the external I/O ports.

Memory reference subcycles are either memory read (data is returned from memory) or memory write (data is stored in memory). Several machine states are usually required to execute a memory reference subcycle. For example, in a memory read subcycle status information is first provided by the control logic, indicating a memory read cycle is to be excecuted. Following status information, the memory address is sent to the external address bus. At this point in the subcycle, depending on the capabilities of the microprocessor, the memory read signal is issued or delayed if necessary to allow for minimum memory access time. Finally, data is returned from memory on the external data bus and routed to the appropriate destination register within the CPU.

For a memory write subcycle, the sequential operation is reversed slightly compared to the memory read subcycle. In this case, the first and second steps are similar; that is, status information is sent out by the control logic indicating a memory write subcycle is to be performed. Then, the memory address is placed on the external memory address bus. The third step differs in that data (instead of a control signal) is sent out by the CPU to the external memory data bus. Finally, the memory write signal is sent out by the control logic. Again, some microprocessors provide a means to delay the memory write signal to allow for minimum memory access time.

Microprocessors providing delay for slow memory or I/O have what is called a ready control signal input and a special machine state called a wait state. As long as the ready control signal input is true, CPU operation takes place at full machine speed set by the CPU master clock. Making the ready input false temporarily freezes CPU operation while retaining all CPU status. Usually, the ready input is checked just prior

to the memory or I/O, read or write control signal. Thus, address and data are held constant while in the wait state. If memory or I/O is too slow when the CPU operates at full speed, a wait-cycle generator can be used to place the CPU in the wait state for a minimum period of time required for memory or I/O access. Memory or I/O that is fast enough to operate with the CPU at full machine speed is said to be capable of operating synchronously with the CPU. Memory or I/O which is not fast enough and requires the CPU to synchronize with the external memory or I/O cycle is called asychronous memory or I/O. Assuming the microprocessor has a ready signal input, slow memory requires an external wait-cycle generator to delay the appropriate read or write control signal.

The class of microprocessors that do not provide a wait state require additional hardware to operate with asynchronous memory or I/O. First, a circuit is needed to recognize the memory or I/O cycle. The problem arises at this point as the only alternative to suspend CPU operation, yet retain data on the address and data buses, is to provide external address and data storage and place the CPU in DMA if indeed DMA is provided. A possible second alternative is to stop the CPU master clock. Stopping the clock, however, is usually not a solution since many processors are dynamic and internal status requires clock refresh cycles. In practice, the clock pulse is stretched just long enough to allow the memory or I/O access. The hardware to provide this operation is considerably more involved than a wait-cycle generator circuit.

Using slow memory or I/O with a microprocessor may be possible without any external wait-cycle generation circuitry. If the application does not require the microprocessor to be run faster than the memory or I/O access times, the CPU clock frequency can be reduced to meet memory or I/O access times. This solution is applicable only if the minimum CPU clock frequency requirement is met. For example, the Pro-Log MPS system used in the UT PDCP (see Section 5.1) uses a $1.6\overline{66}$ μsec state time which allows the use of slow ROM without need for a wait-cycle generator.

Internal processing subcycles generally require the least amount of time (fewest number of states) to execute. Typical internal processing cycles are register-to-register move, arithmetic operations between registers, and increment a register's contents. Since the internal operations involve only internal storage elements, no time is required for access of external I/O or memory. The minimum cycle time specified for a particular machine is usually the minimum time in which some simple internal CPU operation can be performed. Minimum cycle time for a CPU is not necessarily a good criteria for evaluation of a particular microprocessor. Although the minimum cycle time of two microprocessors may be the same, one may perform a larger function than the other during the minimum cycle time.

As presented earlier, some instruction cycles may be a combination of basic I/O, memory reference, or internal CPU operation subcycles. For example, one instruction may involve reading data from memory and adding the data to the internal accumulator register. Usually two machine subcycles are required to perform an add memory to accumulator type instruction. The first machine subcycle would fetch the add memory instruction, decode the instruction, send out the memory address of the data to be added, and fetch the data to an internal temporary storage register. Several machine states would be required to perform this part of the instruction. The second machine subcycle would be an internal CPU operation subcycle which would be to input the accumulator and the temporary storage register contents to the ALU; direct the ALU to add the operands; and finally, place the result in a destination register (usually the accumulator). Many combinations of machine subcycles can be generated to perform various instruction cycles. The programming of machine subcycles comes under the topic of microprogramming which is discussed in Section 2.2.2.3.

Interrupt and hold control signal inputs are provided by most microprocessors. An interrupt, as the name implies, allows temporary interruption of main program processing to allow for execution of some subroutine. The principle advantage of interrupt capability is evident when using a peripheral much slower than the CPU. Suppose output to a

slow peripheral is required. Without interrupt provision, the processor must wait for the slow output device. With interrupt capability, the processor can be continuously processing, and when the slow output device is ready to take data, an interrupt request is sent to the CPU. The output routine to service the interrupt is executed, and the processor returns to the main processing task. The CPU need never wait for a slow peripheral.

The hold signal input provides just the opposite speed difference capability. Suppose an external peripheral is capable of outputting or inputting data faster than the CPU can process or generate data. If the processor provides a hold cycle, data may be passed between memory and the external device at a rate equal to the access time of the memory plus the time required for the CPU to recognize a hold request. When the control logic recognizes a hold request, the address and data buses are placed in the tri-state mode (floating), and a flag is generated by the control logic to indicate to the external circuitry that the address and data buses are now available for use by the external device that requested the CPU hold cycle. The external device may then address memory directly and perform a memory read or write cycle independent of the CPU. This operation is called direct memory access or simply DMA.

In PDCP applications, the data rate between CPU and sensors, or CPU and the platform transmitter, is generally slow compared to the processing speed of the microprocessor. Therefore, in PDCP applications, the DMA or hold capability will probably not be required.

The only remaining topic of the hardware architecture of the microprocessor that must be considered is the problem of system start-up. All microprocessors have the capability to load the program counter with some particular address upon the application of an external start-up signal. This is equivalent to initiation of a program since the program counter is loaded with the starting address of the program. Perhaps the best way to discuss hardware start-up is through examples.

The 8080A microprocessor provides three techniques for initiating a program. The most fundamental of these is application of an external reset signal to the reset control signal input of the 8080A chip. The control logic recognizes a reset request and places zero in the program counter. All other CPU status is unchanged. Thus, a vectored start-up to absolute address zero is performed. The problem with this method is that either location zero must be the starting point of the start-up program or the location must be preloaded with a jump instruction to the desired location. If location zero is ROM, then on power-up, the system may be started immediately with only a reset pushbutton required. If location zero is volatile RAM, location zero must be preloaded. A further complication results if all memory is volatile; in this case, a short boot-strap loader program must be toggled in by hand, and system start-up would include jumping to the boot-strap loader program which loads the system program(s). Usually, the simple boot-strap loader is used to load a more complicated loader with some sort of error checking. The more complicated loader then loads the system program(s) and passes control to the system program(s) at the conclusion of the loading routine. One microprocessor, the RCA CPD 1802 provides a hardware boot-strap. The CPD 1802 provides a special DMA cycle which permits sequential memory loading starting at location zero.

The second and third methods of starting the 8080A series microprocessors involve the interrupt capabilities of the device. There are eight single-bit restart instructions which cause an unconditional call to eight particular locations in memory (locations 0, 10, 20, 30, 40, 50, 60, and 70 octal). That is, the present value of the program counter is saved in the stack, and the program counter is loaded with one of the eight appropriate restart locations. All other CPU status is unaffected. The restart Ø (restart to location Ø) is similar to a reset, only additional hardware is required to jam the interrupt restart instruction onto the data bus. Also, the stack is affected by the restart instruction. The original 8080 provides only the reset and restart interrupt methods of system start-up. The newer 8080A series of microprocessors permit a three-byte instruction interrupt. That is, a three-byte instruction may be jammed onto the data bus during the interrupt. This permits unlimited

vectoring of the program counter since a call or jump instruction can specify a branch to any of the 2^{16} (65,536) possible memory locations. Of course, additional hardware is required to interrupt with three bytes since three bytes must be presented sequentially to the CPU data bus instead of only one.

Most microprocessors provide some sort of reset input that vectors the program counter to a particular location although the location is not always zero. The Motorola 6800 provides an interesting start-up technique which differs from the three methods just described. The desired program starting address is preloaded in the top two locations of memory. The reset signal initiates a special machine cycle which loads the program counter with the address stored in the top two memory locations. The resulting operation is equivalent to an unconditional jump to the memory location specified by the contents of the top two memory locations. Again, the top memory locations must be either non-volatile or pre-programmed.

2.2 MICROCOMPUTER SOFTWARE ASPECTS

A microcomputer must be programmed in order to perform a process control task. In the case of a microprocessorized DCP, the DCP system algorithms must be converted into a set of instructions (the microcomputer program) that can be directly loaded into the system memory. The process of converting system algorithms into machine executable instructions is often called coding.

As depicted in Figure 2.2(1), there are many levels of microcomputer coding. At the top of the scale are compilers and interpreters such as FORTRAN, PL/1, and BASIC. These high-level languages are machine independent in that a program written in any high-level language can be executed on any machine which supports that particular high-level language. For example, one could write a FORTRAN program to run on a DEC PDP-11/45 and also execute the program on an Intel 8080A microcomputer with little or no change in the original FORTRAN source program. Of course the original program would require recompiling using an 8080A FORTRAN compiler.

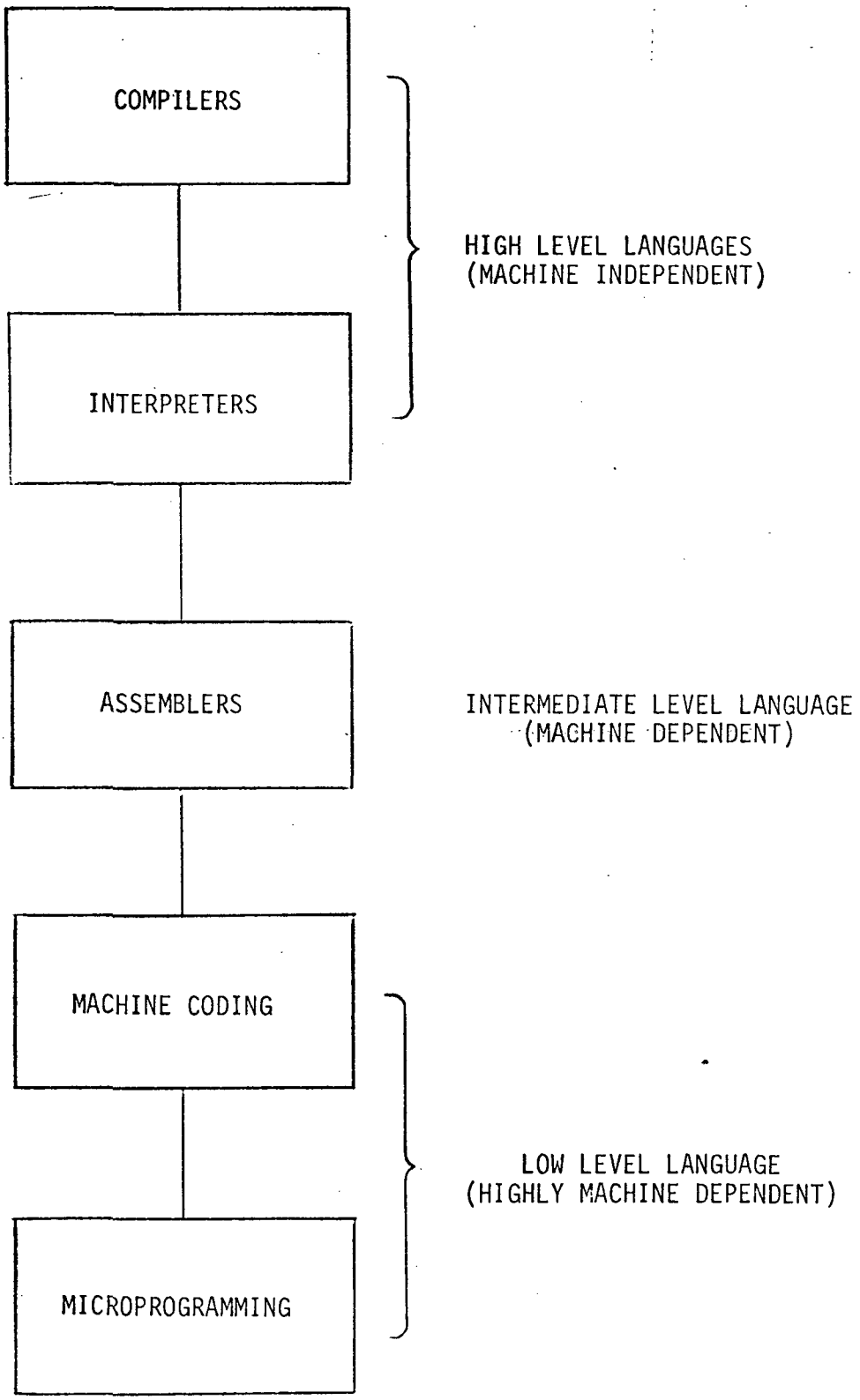


Figure 2.2(1) Coding Levels.

Intermediate-level programming is a more complex process than high-level programming. An assembly language programmer must have a thorough knowledge of the particular microprocessor's instruction set and a complete understanding of the particular microcomputer's hardware architecture. Assembly language programming is called machine dependent programming since instruction sets vary greatly from one microprocessor to another.

At the bottom of the coding scale is low-level programming. Machine language programming is extremely time consuming and should generally be avoided. Machine language programming is the process of hand loading the machine binary digits representing the various instructions the programmer wishes the CPU to perform. The binary digits are literally "toggled in" from a switch register or typed in as numbers from a keyboard. This process requires considerable time and effort from the programmer. Machine coding should not be required for PDCP applications since assembly language programs are available for most current microprocessors. The UT PDCP system described in Section 4 provides machine level coding capabilities as well as simplified symbolic assembly language programming.

The most basic level of microcomputer coding is a highly machine dependent technique called microprogramming. Microprogramming is sometimes referred to as the bridge between hardware and software [8]. This is because a microprogram consists of the control words used to decode machine instructions (software) into machine operations (hardware). The microprogram resides in the control storage element of the CPU. The control storage device is either a read only memory (ROM) or programmable logic array (PLA). Some microprocessor manufacturers allow the user to specify the microprogram. Since the microprogram defines the instruction set of the microprocessor, this allows the user to develop the optimal software structure for a particular application.

In addition to programming languages, the microcomputer programmer requires several utility programs to complete the programming task. A text editor program is useful in preparing the source code of a program.

Source code is the name given to the human readable program. Source code is compiled or assembled into equivalent machine executable object code. Another required utility program is a loader which is used to load the object code output of a compiler, interpreter or assembler program directly into the microcomputer memory. Finally, debugging and simulator programs aid in testing the object program. The remainder of this section provides a more detailed description of the software aspects of the microcomputer.

2.2.1 Microcomputer Instruction Sets

A microcomputer instruction provides the binary information required by the control logic of the CPU to perform a particular processing task. According to Weiss [9], microprocessor instructions can be conveniently grouped into four basic categories:

1. Data Movement
2. Data Manipulation
3. Decision and Control
4. Input/Output

Data movement instructions control the movement of data from register-to-register, register-to-memory, memory-to-register, and memory-to-memory. Figure 2.2.1(1) graphically illustrates possible sources and destinations for data movement. Note that data flow to and from input/output ports is also depicted in Figure 2.2.1(1). Input/output data movement is a special case of the general class of data movement instructions.

Data manipulation instructions provide arithmetic and logical operations on data. Arithmetic instructions include add, subtract, multiply, divide, and increment/decrement. Typical logical operations are logical AND, OR, exclusive OR, complement, compare, and rotate/shift. All microprocessor instruction sets provide the basic binary add instruction. If other data manipulation instructions are required but are not included in the instruction set, the needed instructions can usually be implemented with a subroutine.

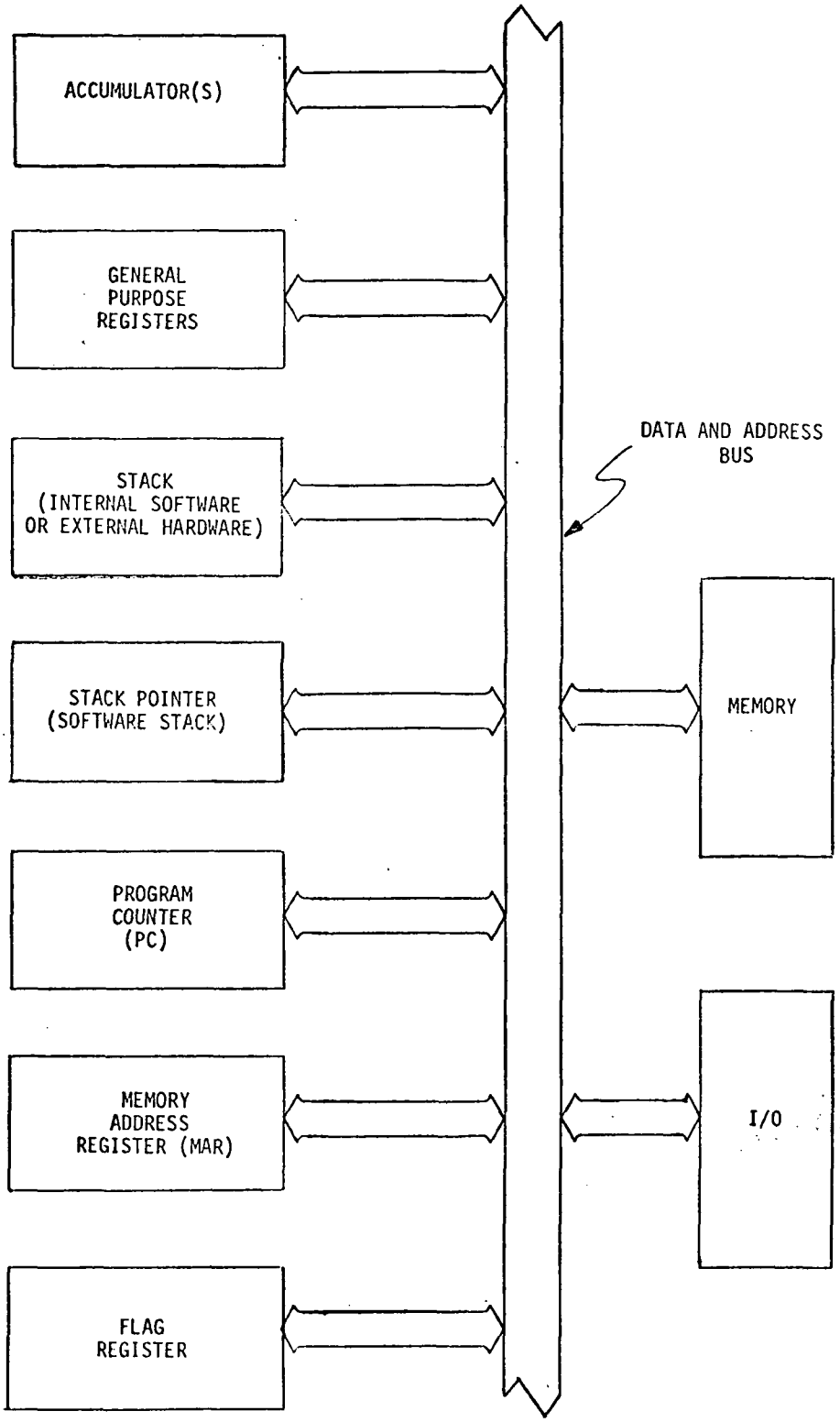


Figure 2.2.1(1) Data Movement in the Microcomputer.

Microprocessors execute instructions in a sequential manner unless otherwise directed by a decision and control instruction. Non-sequential alteration of the program counter is a result of the execution of a conditional or unconditional jump, jump to subroutine, return from subroutine, or skip instruction. Unconditional jump or call instructions are sometimes called branch instructions. Also, some authors prefer the use of the simpler term "call" to indicate a jump to subroutine type instruction.

The choice of input/output device interfacing depends largely on the nature of the input/output instructions provided by the particular microprocessor. Many microprocessors provide no formal input/output instructions. Instead, memory referencing instructions must be used with a technique called memory-mapped I/O (see Section 2.1.3). Input/output instructions provide data transfer between the microcomputer and external I/O devices.

An explanation of the instruction set of a particular microprocessor can generally be found in the user's manual published by that microprocessor's manufacturer. A complete description of the Intel 8080A instruction set used in the UT PDCP development system is contained in the Intel 8080A User's Manual [10]. A copy of the Intel 8080A User's Manual is included with the UT PDCP System Operation Manual.

Microcomputer instructions are similar to typical minicomputer instructions. The major difference is that the execution speed of instructions is generally much faster for a minicomputer. Recent technological advances, however, are narrowing this speed gap. Since the form and resulting operations of microcomputer instructions are similar to minicomputer instructions, texts and papers discussing general minicomputer instruction sets are applicable to microcomputer instructions. References 9, 11, and 12 provide excellent discussions of typical mini- and microcomputer instruction sets.

Microcomputer instructions sets vary considerably from manufacturer to manufacturer. The ability of a particular microprocessor to efficiently implement a PDCP is primarily a function of that particular

microprocessor's instruction set. Evaluation of the microprocessor's instruction set must be a primary consideration in the selection of a particular microprocessor for a PDCP design. As a result of the software development incorporated in this study (see Section 4), certain types of instructions appear to be highly desirable for PDCP applications. Choosing a microprocessor that contains as many of the desirable instruction types as possible will result in reduced memory costs due to more efficient coding. These desired instruction types are given higher weighting factors in the microprocessor evaluation presented in Section 6 of this report.

2.2.2 Microcomputer Programming

As indicated at the beginning of Section 2.2, the procedure for converting process control system algorithms to machine executable instructions is often called coding. Efficient coding requires a well structured algorithm. Flowcharting is a useful technique for reducing the algorithm to a form which can easily be converted to a computer program. The flow chart is a graphical representation of the distinct operations required to implement the computer program. Numerous examples of program flowcharting appear throughout Section 4 of this report. Typical flow chart symbols are presented in Figure 2.2.2(1). Once the flow chart for a program is developed, the programmer proceeds to implement each block of the flow chart using an appropriate programming language.

The basic programming tools available to the microcomputer programmer are:

1. Programming languages
2. Utility programs
3. Microprogramming

The first two tools are programs that are actually run on the particular microcomputer (resident program) or on some other computer (cross programming). Programming languages and utility programs are discussed in

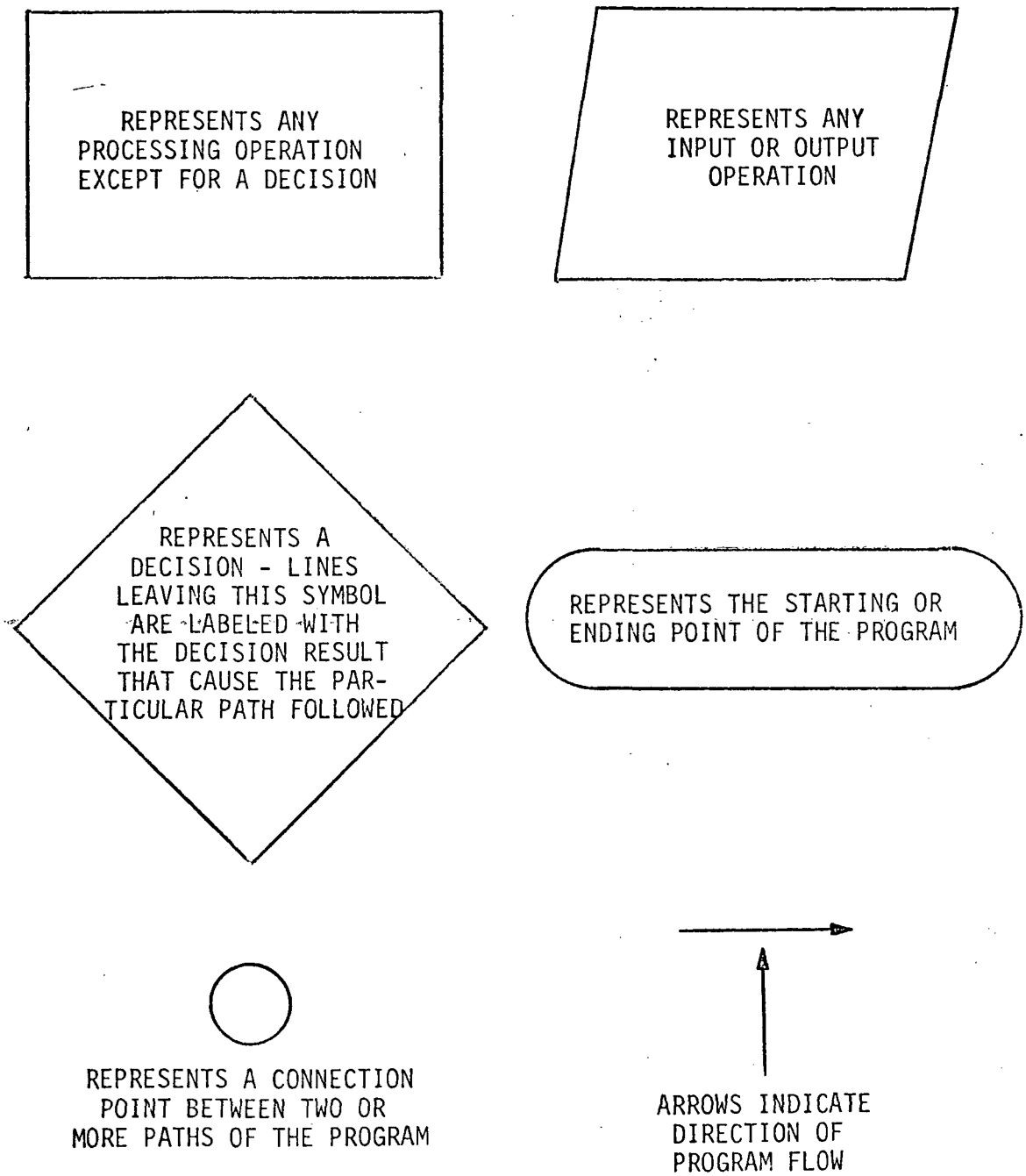


Figure 2.2.2(1) Typical Flow Chart Symbols.

Sections 2.2.2.1 and 2.2.2.2, respectively. The third tool, microprogramming, is discussed in Section 2.2.2.3 of this report.

2.2.2.1 Programming Languages - Programming languages are conveniently grouped into three classes [see Figure 2.2(1)]:

1. High-level languages (compilers and interpreters)

2. Intermediate level languages (assemblers)

3. Low-level languages (machine language and microprogramming)

High-level languages include compilers and interpreters such as FORTRAN, PL/M, COBOL, FOCAL, or BASIC. All five of these languages are currently available for one or more microprocessors. For example, the Intersil 6100 will support all software available for the DEC PDP-8E minicomputer including FOCAL, BASIC, COBOL, and FORTRAN. The Intel 8080A used in the UT PDCP development system is supported by FORTRAN, PL/M, BASIC, FOCAL, and COBOL. The significant advantages of high-level programming are machine independence and a reduction in programming time. High-level languages can reduce programming time by 50 to 80 percent. Also, existing high-level language programs can be recompiled to run on a microprocessor which supports the particular high-level language used. The high-level programming languages were each developed for particular programming tasks. FORTRAN is designed primarily to solve mathematical problems and thus may be useful for PDCP data processing. COBOL, however, is a business oriented language and would have little application in the PDCP field. PL/M, a microprocessor compiler originated by Intel, is a subset of IBM's powerful PL/1 compiler. PL/M provides instructions oriented toward commercial control and scientific problem solving. BASIC is an interpretive language in that each line of source code is compiled as the program executes. BASIC requires a relatively large amount of memory to execute even short programs. This is because the BASIC source program and the BASIC interpreter program must both be resident in memory during program execution. A compiler differs from an interpreter in that the final compiled program is machine executable object code. The compiler is not needed once the object code is

generated. Finally, FOCAL is a language which was developed by Digital Equipment Corporation to fill the gap between BASIC and FORTRAN. Like BASIC, FOCAL is an interpretive language that requires considerable memory overhead to run even short programs. Neither BASIC nor FOCAL are recommended for potential PDCP application due to their high memory requirements.

The principle disadvantage of using any high-level language for PDCP programming is the inefficiency of the compiled object code. Another significant disadvantage is inadequate input/output flexibility. Also, program execution times are generally unknown and uncontrollable. PDCP software timing routines require precise knowledge of the number of machine states required to execute various subroutines. Even interrupt timing techniques (see Section 4) require knowledge of execution times. In this case, worst case execution times must be known to guarantee that the subroutine will be completed within the alotted time span.

Assembly languages are intermediate level languages. An assembler translates symbolic machine instructions or mnemonics directly into machine executable object code. A mnemonic is a short form symbolic name given to each instruction in the instruction set of a particular microprocessor. Mnemonics used to represent instructions vary greatly from manufacturer to manufacturer. This is due to the wide variation in microprocessor instruction sets. Since microprocessor instruction sets vary from machine to machine, assembly language programming is machine dependent. A primary disadvantage of assembly language programming is a direct result of the variation of instruction sets. The microcomputer programmer must be thoroughly familiar with the instruction set of the machine to be programmed. Furthermore, efficient assembly language programming requires complete understanding of the particular microcomputer's hardware architecture.

Many types of microcomputer assemblers are available. Cross assemblers are run on a machine other than the microcomputer for which the program is written, whereas resident assemblers are run on the machine for which the program is written. All assemblers translate symbolic

instructions to their equivalent object code. Several additional features are often found on current microcomputer assemblers. The most useful feature is a provision for symbolic addresses and constants. These symbolic names and labels can be used to represent address and data constants. This feature reduces programming errors by freeing the programmer from the burden of keeping track of absolute addresses and constants. Some assemblers even provide algebraic manipulation of address and data expressions. Recognition of pseudo assembly directives is provided by most assemblers. For example, the pseudo directive "END" informs a typical 8080A assembler that the location of the END directive marks the end of the source code to be assembled. Macroassemblers usually provide for all the above features plus the assembly of macroinstructions. A macroinstruction is a single line instruction used to represent a multi-instruction sequence. This feature supplies some of the programming simplicity of a high-level language since a single line of source code can replace many machine instructions. Note, however, that assembly language efficiency is retained. Other features provided by some microcomputer assemblers include conditional assembly directives, relocation and linkage of multiple program segments, optional assembler listing formats, and loading of object code output directly into memory or to some external storage device for later loading into memory. Microcomputer assemblers are quite similar to typical minicomputer assemblers. References 13 and 14 provide general discussions of typical mini- and microcomputer assemblers.

2.2.2.2 Utility Programs - Once a program has been written and translated to object code, the object code must be loaded into the system memory and the program must be checked for correct operation. A loader program is used to load the object code directly into memory. A loader program is not required when using a programming language that outputs object code directly to memory. This feature is provided by some microprocessor assemblers. Certain high-level languages do not have object code output (such as BASIC). In this case, a loader is required to enter the source code. Several types of program loaders are available for current microcomputers. The simplest is a boot-strap loader. The boot-strap loader program is purposely very short and

provides none of the error checking or other advanced features typical of the more complex general purpose loaders. A simple boot-strap loader could be used to load any program; however, the boot-strap loader is typically used to read a more sophisticated loader. Control is then transfered to the second loader which usually provides error-checking and automatic program start-up following the loading process. Unless a machine has non-volatile memory pre-programmed with a loader, the programmer must manually load a boot-strap loader program to initiate subsequent program loading. The UT PDCP development system includes an error-checking cassette loader routine as an integral part of the system monitor program (see the System Operation Manual). The loader routine is stored in PROM (programmable read only memory) and is available to the user at all times.

Once a user program is loaded into memory, the program can be executed and checked for correct operation. If the program runs correctly, the software task is complete. Typically, however, programs contain "bugs" which prevent correct operation of the program. The bugs must be removed by a technique called program debugging. Microcomputer programs which allow examination and alteration of memory contents to assist the programmer in debugging and correcting the program are available. Often these programs include a routine which will print the CPU status (register contents). Breakpoints may be set in memory to temporarily suspend program execution when a specified point in the program is encountered. Thus, breakpoints permit the programmer to examine CPU status and memory at any point of program execution. This enables the programmer to debug the program in small sections. As bugs are found, they are corrected, if possible, by altering the instruction sequence in memory. Often the programmer will initially insert "no operation" instructions throughout an untested program. This provides memory space between the original program instructions so that additional instructions can be inserted if they are required. The UT PDCP system monitor contains extensive program debugging aids.

A simulator program provides a very powerful technique for testing and debugging a microcomputer program. A simulator program, as the name

implies, simulates the operation of the microcomputer. Program development and testing is simplified by using a large computer system supported by peripherals such as a video display terminal, a high-speed line printer, and a disk drive. The full potential of a simulator can only be realized by using such a developmental system.

The simulator provides the basic function of program instruction tracing. That is, the result of executing each instruction can be displayed and verified by the programmer. Any discrepancy between what the program actually does and what the programmer feels should be happening is immediately evident. The programmer can then patch the program to correct errors as they are found. The process continues until the program runs without error. Some simulators provide additional features such as counting machine states required to execute a program segment, optional tracing and listing modes, and simulated memory examination and alteration. Final program verification must be performed on the actual PDCP system since the simulator cannot verify correct operation of I/O and control interfacing. For example, external A/D conversion may be dependent on software timing which must be verified on the actual PDCP hardware system. Full simulator potential plus final checkout could be achieved simultaneously by using a microcomputer development system that includes the same hardware interfaces that are used on the PDCP system.

2.2.2.3 Microprogramming - A microprogram is an integral part of the control logic of most microprocessors. In a microprogrammed control unit, the numerous individual operations required to execute each instruction cycle are defined by microinstructions fetched from a read-only-memory or a programmable logic array. Therefore, each macroinstruction written by a user is actually executed by a microprogram, and the instruction set of the microprocessor is defined by the set of microprograms stored in the CPU control unit read-only-memory.

Some microprocessor organizations are designed to permit users to specify the microprogram. These microprogrammable microprocessors offer users the potential advantage of defining an optimal instruction set for their particular application. Microprogramming essentially extends the advantages of programmed logic one step further down the quantum ladder.

The constraint of a predefined instruction set is removed leaving only the basic constraints imposed by the hardware architecture of a particular microprocessor.

The PDCP software development portion of this study (Section 4) provides some insight into the problem of defining an optimal microcomputer instruction set for a PDCP. In Section 4.3, the potential contributions of microprogramming to the development of block operator subroutines for the library of an applications-oriented software development system are outlined. General microprogramming concepts are discussed in References 8, 15, 16, 17, 18, and 19.

REFERENCES

1. Lewis, D. R., "Microprocessor or Random Logic," Electronics Design 18, September 1, 1973.

2. Electronics Staff, "Diverse Industry Users Clamber Aboard the Microprocessor Band Wagon," Electronics, July 11, 1974, pp. 81-108.

3. Altman, L., "Single-Chip Microprocessors Open Up New World of Applications," Electronics, April 18, 1974, pp. 81-87.

4. Smith, H., "Impact of Microcomputers on the Designer," 1973 WESCON Technical Papers, Session 2.

5. Morris, J. H., Patel, H., and Schwartz, M., "Scamp Microprocessor Aims to Replace Mechanical Logic," Electronics, September 18, 1975, pp. 81-87.

6. Chung, D., "Four-Chip Microprocessor Family Reduces System Parts Counts," Electronics, March 6, 1975, pp. 87-92.

7. Weissberger, A. J., "Microprocessors Simplify Industrial Control," Electronic Design 22, October 25, 1975, pp. 96-99.

8. Galey, J. M., "Microprogramming: The Bridge Between Hardware and Software," Computer, August 1975, Vol. 8, No. 8, p. 23.

9. Weiss, C. D., "Software for MOS/LSI Microprocessors," Electronic Design 7, April 1, 1974, pp. 50-57.

10. Intel 8080 Microcomputer User's Manual, July 1975, pp. 4-1 to 4-15.

11. Mano, M. M., Computer Logic Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 326-377.

12. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, pp. 44-63.

13. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, pp. 206-213.

14. Weiss, C. D., "MOS/LSI Microcomputer Coding," Electronic Design 8, April 12, 1974, pp. 66-71

15. Jones, L. H., "A Survey of Current Work in Microprogramming," Computer, August 1975, Vol. 8, No. 8, pp. 33-37.

16. Broadbent, J. K., "Microprogramming and System Architecture," The Computer Journal, Vol. 17, 1974, pp. 2-8.

17. Cox, G. W. and Schneider, V. B., "On Improving Operating System Efficiency Through Use of a Microprogrammed, Low-Level Environment," Preprints, Micro 7, pp. 297-298.

18. Montangero, C., "An Approach to the Optimal Specification of Read-Only Memories in Microprogrammed Digital Computers," IEEE - Trans. Computers, Vol. C-23, April 1974, pp. 375-387.

19. Vandling, G. C. and Walddecker, D. E., "The Microprogram Control Technique for Digital Logic Design," Computer Design, August 1969, pp. 44-51.

3. PDCP SYSTEM HARDWARE

PDCP system hardware must be designed to satisfy the constraints imposed by the remote data collection application. Typically, design goals include minimizing power consumption, weight, maintenance, and cost while providing accurate and reliable operation. As illustrated in Figure 3(1), DCP hardware can be partitioned into three systems consisting of the sensors, the digital control logic, and the transmitter. This section discusses the influence of microprocessor technology on the design of each system of a PDCP. In addition, standards which could promote lower PDCP costs are recommended, and the development of a standard, modular PDCP design is proposed.

3.1 SENSOR CLASSIFICATION AND INTERFACING

The most variable elements in data collection systems are the sensors. With measurements being made in such diverse fields as agriculture, ecology, hydrology, and search and rescue, many types of sensors must be accommodated by the data collection platform. A signal conditioner is generally placed between the sensor output and the DCP to convert the sensor output to a standard electrical signal which is compatible with the DCP. In previous systems, this signal has been a voltage, a frequency, or a digital logic level depending upon the particular design of a data collection platform telemetry system. For programmable data collection platforms, the ideal interface occurs at digital logic levels. For analog sensors, an analog-to-digital converter is required as a signal conditioner. The sensor interface is then made at the input to the analog-to-digital converter.

3.1.1 Sensor Classification

User requirements have been documented in recent studies [1]. Table 3.1.1(1) gives a list of the disciplines which have a need for data collection by satellite. Table 3.1.1(2) provides a list of parameters for which sensors are required and the discipline in which the

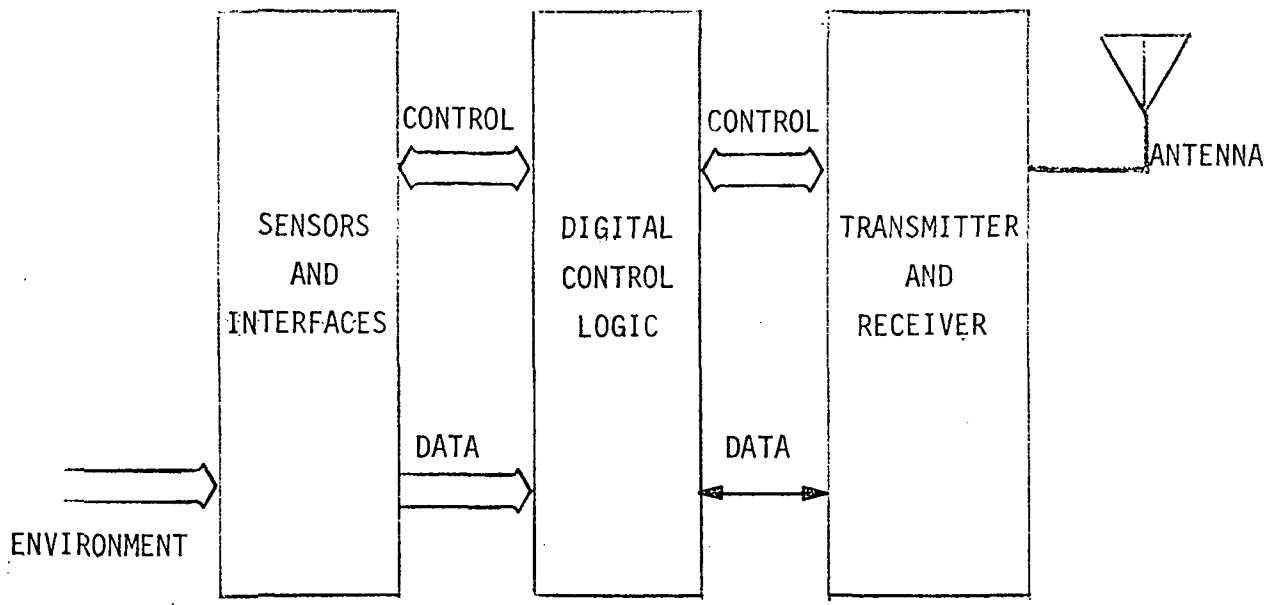


Figure 3(1) DCP Hardware Subsystems.

TABLE 3.1.1(1)

SATELLITE DATA COLLECTION DISCIPLINES

Disciplines	
Agriculture	Meteorology
Biological Behavior	Navigation
Ecology	Oceanography
Geology	Search and Rescue
Hydrology	Transportation

TABLE 3.1.1(2)

SENSOR MEASUREMENT PARAMETERS AND
ASSOCIATED DISCIPLINES

Parameter	Discipline
Acceleration	Biological Behavior
Acoustic Noise	Oceanography
Acoustic Output	Biological Behavior
Activity (Time Up and Down)	Biological Behavior
Aerosols	Meteorology
Air Particles	Biological Behavior Ecology
Air Pollution	Agriculture
Albedo	Geology
Bioluminescence	Oceanography
Blood Pressure	Biological Behavior
Body Position	Biological Behavior
Cargo Security	Transportation
Chlorine	Oceanography
Cloud Cover	Agriculture
Conductivity	Ecology Geology
Creepmeter	Geology
Crop Condition	Agriculture
Dew Point/Front Point	Hydrology
Dissolved Oxygen (Water)	Ecology Hydrology
Diving Depth	Biological Behavior
EKG	Biological Behavior Ecology
Evaporation	Hydrology
Geographical Fix	Agriculture Biological Behavior Ecology Geology Oceanography Search and Rescue Transportation

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Heading (Compass)	Biological Behavior
Heart Rate	Biological Behavior
Humidity (Relative)	Agriculture Ecology Geology Meteorology Oceanography
Ice Presence	Hydrology
Ice Thickness	Hydrology
Light at Surface	Ecology
Light Penetration	Ecology
Magnetic Field	Geology
Oxidents	Meteorology
Oxygen Concentration	Agriculture
Particle Counts (Airborne)	Ecology
pH (Soil)	Ecology
pH (Stomach)	Biological Behavior
pH (Water)	Ecology Hydrology
Phytoplankton Counting	Ecology Oceanography
Pipeline Current	Geology
Pipeline Voltage	Geology
Pitch Angle	Biological Behavior
Pore Pressure	Geology
Precipitation	Agriculture Ecology Hydrology Meteorology
Pressure, Atmospheric	Meteorology Oceanography
River Stage	Ecology
Salinity	Agriculture Ecology Oceanography

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Sediment	Ecology
Seismometer	Geology
Snow Cover	Agriculture Meteorology
Snow Depth	Agriculture Hydrology
Snow Moisture Equivalent	Hydrology
Soil Moisture	Agriculture Ecology Hydrology
Solar Radiation (Surface and Depth)	Ecology
Solar Radiation (Net Allware)	Agriculture Biological Behavior Ecology Hydrology Meteorology
Specific Conductance	Hydrology
Storm Surge	Meteorology
Strain, Biaxial	Geology
Strain Gage	Oceanography
Speed, Swimming	Biological Behavior
Tail Beat (EKL)	Biological Behavior
Telluric Current (On Pipe)	Geology
Telluric Current Frequency	Geology
Temperatire, Air	Agriculture Biological Behavior Ecology Geology Meteorology Oceanography
Temperature, Body Surface	Biological Behavior
Temperature, Cargo	Transportation
Temperature, Deep Body	Biological Behavior

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Temperature, Sea Surface	Meteorology
Temperature, Soil	Ecology Meteorology
Temperature, Stomach	Biological Behavior
Temperature, Surface	Biological Behavior
Temperature, Water	Biological Behavior Ecology Hydrology Oceanography
Tide Height	Ecology
Tilt, Biaxial	Geology
Time	Biological Behavior
Time In and Out of Water	Biological Behavior
Total Organic Carbon	Ecology
Tsunami	Geology
Turbidity	Ecology Hydrology Oceanography
Vapor Pressure	Agriculture
Water Current Profile	Oceanography
Water Depth	Biological Behavior
Water Direction	Oceanography
Water Level	Geology Hydrology
Water Table Depth	Hydrology
Water Velocity	Biological Behavior Ecology Hydrology Oceanography
Wave Height	Transportation
Wave Spectra	Oceanography
Wildland Fire	Ecology
Wind Profile	Agriculture

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Wind Speed	Biological Behavior Ecology Geology Hydrology Meteorology Oceanography
Wind Vector (Direction)	Agriculture Hydrology Meteorology Oceanography

sensor could be utilized. Among this list are sensors which have already seen service in satellite data collection systems. A typical group composed of hydrology sensors is listed in Table 3.1.1(3) along with their signal conditioner outputs. While the direct output of each sensor may be a measurement in volts, amperes, or ohms with nonlinear and suppressed zero scales, the signal conditioners have conveniently translated the output ranges to lie between zero and plus five volts.

3.1.2 Sensor Interfacing

A number of sensors already exist which could interface directly with a programmable data collection platform including those in Table 3.1.1(3) with the addition of an analog-to-digital converter. Some sensors include their own signal conditioning equipment which converts the basic sensor output to a digital logic level. For example, the temperature sensor in the TWERLE experiment converts the measured temperature to a frequency output. A counter measures the number of cycles of this frequency over a fixed period; the contents of the counter registers are a digital representation of the temperature value.

Two basic classes of signal conditioners will probably be required in a PDCP system. One class converts the non-standard basic sensor output to a standard output. This will normally be the responsibility of the sensor designer. A second class of signal conditioners converts the standard output to digital logic levels which are compatible with the microcomputer. In many cases, the second class of signal conditioners will be time-shared between a number of sensors and would be provided as a standard module by the PDCP designer. Of course, a direct conversion of the basic sensor output to a logic level is very desirable since this eliminates the need for the second class of signal conditioners; however, the second class of signal conditioners will be required to utilize existing sensors. Fortunately, these signal conditioners convert from a standard input to a standard output; therefore, their cost is low.

TABLE 3.1.1(3)

TYPICAL SATELLITE DATA COLLECTION SENSORS

Sensor	Signal Conditioning Output Specification
Dissolved Oxygen	0.0 to +5.0 Volts dc
pH	0.0 to +5.0 Volts dc
Specific Conductance	0.0 to +5.0 Volts dc
Temperature	0.0 to +5.0 Volts dc

3.1.2.1. Sensor Interface - Little can be said about the basic sensor output since this can be almost any variable. The sensor signal conditioner takes this output and converts it to a standard output. Traditionally, for analog sensors this output has been zero to plus five volts dc. Another standard range for low-voltage or low-current devices has been zero to 50 millivolts dc. If at this point in time a recommendation were to be made for a standard output range for analog sensors, it would have to be zero to plus five volts. This, however, may present a problem if some of the newer technologies are used in fabricating the microprocessor. The integrated injection logic (I^2L) technology (see Section 6.1) can operate at voltages lower than one volt, and operating the signal conditioner from the same power supply voltage would be advantageous. This would require a low-voltage output from the signal conditioner.

As previously mentioned, some sensors produce a digital output directly. The analog-to-digital conversion process may be built into the sensor. The output of such a sensor should have an interface compatible with CMOS logic. In contrast to the analog output of zero to plus five volts dc, the digital sensor output should be as follows: a zero would be represented by a level between zero and 0.8 volts, and a logical one would be represented by a level of 3.5 to 5 volts dc. Until more experience is gained with I^2L devices, the above specification for the output of the digital sensor would be a reasonable recommendation.

3.1.2.2. Microprocessor Interface - The second level of signal conditioning converts the standard interface signals to signals which match the logic characteristics of the microprocessor. The most common signal conditioner of this type is the analog-to-digital converter (ADC). This device takes a zero to plus five-volt dc analog signal and converts it to an equivalent digital word. For the PDCP application, the ADC output should be coded in natural binary at CMOS compatible logic levels. A parallel output word will generally be required for direct interface to the microprocessor data bus. However, in some applications where the ADC and sensor must be located in an area remote from the PDCP a clocked serial output is desirable to minimize the number of wires required for the data link. Both the parallel and serial outputs should be three-state outputs to permit the signal lines to be shared with other devices.

In a three-state device two states are the usual high or low output (input) states. A control signal converts this output (input) to a high impedance state effectively removing the device from the bus. Suitable low-power CMOS-logic ADC's which would interface directly to the bus of a CMOS microprocessor-based PDCP are commercially available.

Similar to the analog-to-digital converter is the frequency-to-digital converter which is needed by sensors having an output frequency proportional to the measured variable. This conversion of frequency to a digital signal can be performed by the microprocessor with no external hardware. Using the Intel 8080A-1 microprocessor, sensor frequencies up to approximately 20 kilohertz can be digitized. The frequency-to-digital conversion capability of the microprocessor provides a low cost interface to a variety of sensors since most sensor outputs can conveniently be produced as a frequency.

A big advantage in interfacing the microprocessor with a set of sensors is that sensors with different resolution requirements can easily be accommodated. For example, one sensor may require a resolution of eight bits and another require a resolution of sixteen bits from a time-shared ADC. For the eight-bit resolution sensor only the eight highest-order bits need to be multiplexed from the ADC while the whole sixteen bits of the high resolution sensor would be transferred to the microprocessor. This is easily accomplished through program control.

3.2 DIGITAL CONTROL LOGIC AND MEMORY

In current DCP designs, the digital control system is generally a hardwired logic circuit manufactured with standard CMOS integrated circuits. Because of the cost and complexity of the random logic design, control functions are limited to elementary sensor control, data formatting, and transmitter control. A microprocessor-based PDCP will be capable of performing additional tasks such as data compaction and data preprocessing without excessive cost increases (see Section 4).

A block diagram of a microprocessor-based PDCP control system is illustrated in Figure 3.2(1). The number of integrated circuits required to implement this system will depend upon input/output require-

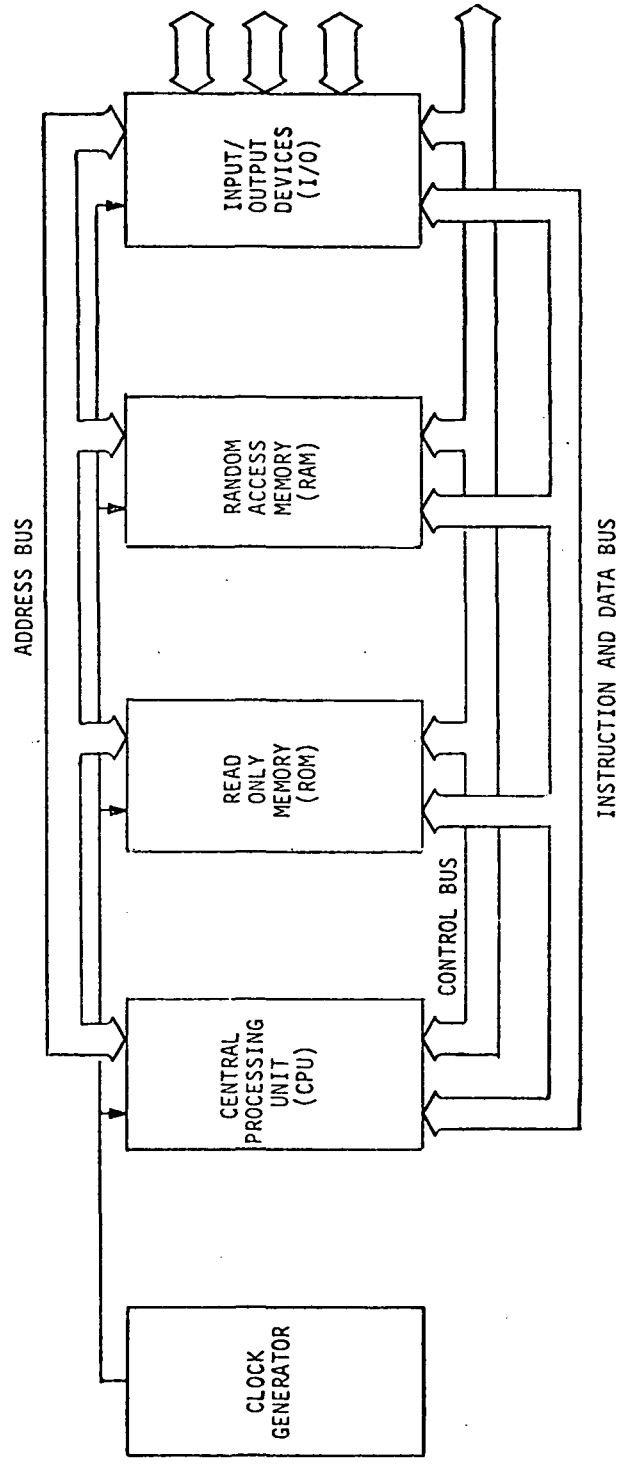


Figure 3.2(1) Microprocessor Based PDCP Control Subsystem Organization.

ments, memory requirements, and the microprocessor architecture. Already, two-chip microprocessor sets containing a clock circuit, 64 bytes of RAM, 1024 bytes of ROM, and several input/output ports are available. Thus, a minimal PDCP control system may require as few as two integrated circuits.

Future DCP user requirements are expected to range from basic platforms capable of performing the tasks of current DCP's through advanced platforms capable of extensive data preprocessing and data compaction. From the standpoint of economics, a standard, universal DCP design is desirable. This goal is impractical for a random logic design but is practical for a microprocessor-based PDCP design. As illustrated in Figure 3.2(1), the microprocessor-based PDCP control system organization is based upon address, control, and data buses. This organization produces a system which is modularly expandable. A basic PDCP capable of replacing current DCP designs will utilize all of the functions shown in Figure 3.2(1). An advanced PDCP with extensive data compaction capabilities will utilize the identical functions. The distinction between these systems will be the amount of ROM, RAM, and I/O provided and their software organizations. Section 4 demonstrates that a unique PDCP software organization can be defined for each user by plugging a different ROM into the system. Thus, a PDCP design which will function efficiently as a basic DCP but which can easily be expanded to form an advanced processing PDCP can be developed.

Physically, the standard PDCP design could be implemented as a modular set of printed circuit (PC) boards. A basic PDCP system would probably require one or two small PC boards. The system could be expanded by adding extra integrated circuits in spaces provided on the basic PC card set. Large scale expansion would involve adding standard memory or I/O cards and, possibly, bus driver circuits. This implementation of the standard PDCP design would provide most potential users with the capability they require at a minimum cost since the design and development expenses could be prorated. If a potential user did require a unique PDCP construction, the electrical design would not necessarily have to be changed. Additional PC board development cost would be incurred, but additional circuit design cost would not be incurred.

3.3 TRANSMITTER, RECEIVER, AND ANTENNA

Section 4.4.3 demonstrates that a microprocessor-based PDCP can easily control transmitter modules which have been developed for current DCP designs. Transmitter control inputs should be digital signals compatible with the technology chosen for the PDCP control system. Otherwise, no constraints are imposed upon the electrical design of transmitters and antennas for PDCP's. Traditional design techniques are satisfactory.

Since the sensor interface and digital control systems of a PDCP will be developed as modular PC board sets, the transmitter system could be similarly designed. One approach is to provide individual transmitter modules designed specifically for each data transmission technique. The PDCP user could then customize the modular system by selecting a transmitter module (Landsat, GOES, TWERLE, or TIROS-N), a basic or expanded control module, and various sensor interface modules. For data collection platforms installed near land lines, a telephone modem module could be chosen instead of a specific transmitter module.

A better approach would be to utilize the flexible nature of the microprocessor to control a single modulator which can emulate the modulation of all of the data collection systems. A standard PDCP system will be capable of operating with all current data transmission formats under software control by using the techniques demonstrated in Section 4.4.3. As a result, a universal transmitter design that can switch between the various data transmission modes under software control could be designed. The required carrier frequencies might be developed from a standard frequency reference by a phase-locked loop containing a programmable frequency divider. This system would eliminate the requirement for multiple transmitter modules and would be more adaptable to any future changes in data transmission techniques. A further study will be required to determine the potential cost effectiveness and performance of the two transmitter construction techniques outlined above.

Some DCP's have been developed with receivers for remote command applications. A PDCP with this capability can be produced by adding a receiver module to the system. The receiver should be interfaced to the PDCP buses at standard logic levels. PDCP software can be developed to poll the receiver and decode the command signals as they are received. Alternately, the receiver can provide a signal to the interrupt line of the microprocessor to indicate that a command signal is being received. The microprocessor would then suspend data processing operations temporarily to input and decode the control message.

3.4 SUMMARY

The problem of standardization of sensor output and microprocessor input is a difficult one to solve; however, attempts should be made towards standardization to reduce the types of signal conditioners required.

It is recommended that new designs for analog sensors have outputs which are 0.0 to +5.0 volts dc. The recommendation for digital interface for CMOS logic is zero to 0.8 volts for a logical zero and 3.5 to 5 volts dc for a logical one.

REFERENCES

1. Wolfe, E. A., Cote, C. E., and Painter, J. E., Satellite Data Collection User Requirements Workship, NASA - Goddard Space Flight Center, Greenbelt, Maryland, May 21, 1975.

4. PDCP SYSTEM SOFTWARE

As demonstrated in Section 2, significant advantages can often be obtained by replacing a hardwired circuit with a software-controlled microprocessor system. The potential advantages of a programmed logic system arise from substitution of relatively inexpensive software storage for complex hardwired circuits. Therefore, the flexibility and cost effectiveness of a PDCP will be highly dependent upon the system software and the extent to which PDCP functions can be software controlled. Unfortunately, system constraints can preclude the use of programmed logic, and, as shown in Section 2, the cost of developing special purpose software for low quantity applications can be excessive. This section will demonstrate that microprocessors are capable of performing PDCP tasks and describe a potential PDCP software package organization designed to minimize software development cost. Included also are discussions of the effects of various PDCP system constraints on the software package and descriptions of subroutines which have been developed for typical PDCP applications.

4.1 PDCP SOFTWARE ORGANIZATION

Standardization is the key to minimum PDCP system cost. However, a PDCP design must be sufficiently flexible to operate efficiently with a number of different sensors, data compaction techniques, and data transmission formats. In addition, a software system organization which provides a simple technique by which individual experimenters and organizations can create special purpose PDCP programs is desirable. The PDCP software system organization outlined in Figure 4.1(1) is designed to meet these goals.

A PDCP software package based on the organization proposed in Figure 4.1(1) would contain an executive program and a number of called subroutines. The executive program controls the operation of the PDCP by executing a set of conditional and unconditional calls to various subroutines. A different executive routine will generally be required for

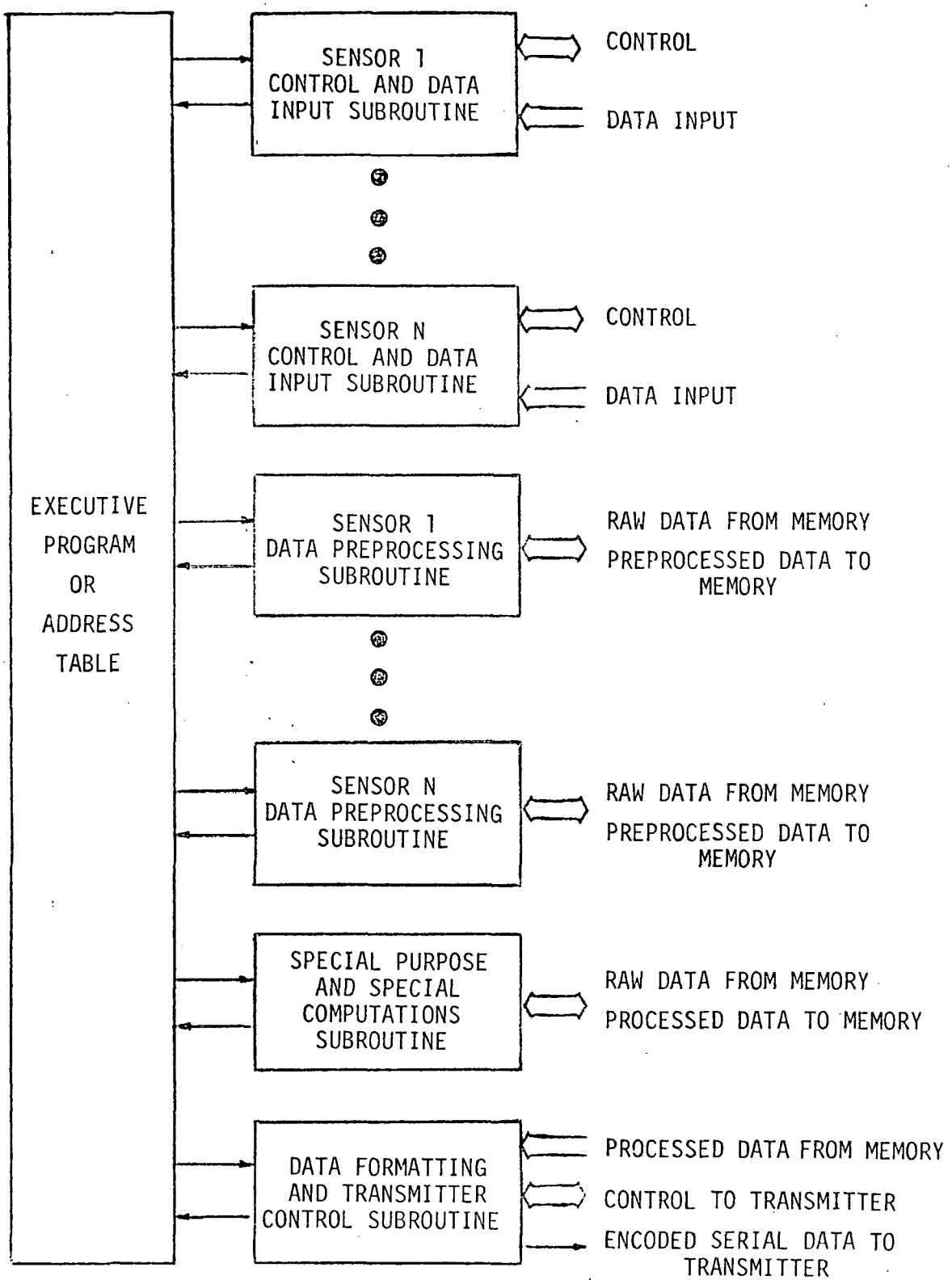


Figure 4.1(1) PDCP Software System Organization.

each different application. Major subroutines called by the executive program control the sampling of a raw sensor data, preprocessing of raw data, special computations, and data transmission to the satellite. The executive program and major subroutines share multipurpose minor subroutines such as software delay loops. Standard sensor data input, general purpose data preprocessing, and transmitter control routines can be provided to users as firmware ROM options for the PDCP. Using this approach, most PDCP software development costs can be prorated over a large number of PDCP's so that the effective software cost will be minimal. The additional program development required for new PDCP software systems could generally be restricted to an executive routine and, when necessary, special purpose subroutines. Since complex control and data processing tasks would generally be controlled by subroutines, the cost of developing executive programs should be low. Special subroutine development costs can be minimized by maintaining a user program library.

4.2 PDCP SYSTEM TIMING

The program organization proposed above is independent of the various system constraints imposed upon PDCP software and the actual techniques used for program development. The major PDCP system constraints which will affect software development is timing. In many cases, the usual goal of minimum execution time will apply to data preprocessing and special purpose subroutines. In general, however, sensor control and transmitter subroutines will require precise timing between specified events, and the total elapsed time between data transmissions should be precisely controlled. These constraints must be considered during PDCP software system development.

Software and interrupt timing are the two basic techniques which can be employed to insure correct PDCP transmission intervals. The microprocessor and associated support logic must operate continuously if software timing is employed. For low power CMOS or I^2L devices, this is not necessarily a disadvantage since maximum throughput can be obtained with a continuously operating system. In addition, however, the precise execution time for each PDCP software routine must be known, and provision must be made for holding the total program execution time between

transmissions constant. Most programs contain various mutually exclusive branches which may require different execution times. As a result, delay must be inserted within certain program branches to equalize execution times. Alternately, program execution time can be allowed to vary if each time the subroutine is called a flag is set to indicate the actual execution time. The executive program may then use any excess time to call optional subroutines (for example, system maintenance or low priority data input routines) and a variable delay subroutine. Although average throughput can be increased using this technique, worst-case throughput is not necessarily improved. The major advantage of software timing is that neither an interval timer nor interrupt logic is required.

Interrupt timing can be relatively independent of system software but requires the use of an external interval timer and interrupt logic. The microprocessor can be powered-down between transmissions or operated continuously. Worst-case execution time for each subroutine must be known in order to guarantee that the microprocessor will either complete data input or processing before a new transmission is required or will be executing an interruptable routine. In the event that the transmission request interrupt is allowed to occur before data processing has been completed, the programmer will have the choice of discarding unprocessed data or completing the processing at the conclusion of the transmission sequence.

Assembly language programming can be used to develop subroutines utilizing either interrupt or software timing. Programs written in assembly language generally provide maximum performance while requiring minimal storage area. Efficient assembly language programming, however, requires a thorough knowledge of the available instruction set and entails the highest development cost per program. Subroutines which require precise internal timing, and general purpose subroutines which will be included in firmware ROM packages provided to PDCP users should be written in assembly language. This will insure precise timing and minimal program storage area. Unfortunately, assembly language programming cost for small-quantity, special-purpose systems could be excessive and should be avoided when possible.

The development of many types of numeric data processing subroutines could be simplified by using a high-level language such as PL-1, FORTRAN, COBAL, or BASIC. Each of these languages is represented by a compiler written for at least one microprocessor. The major disadvantages of these languages are inadequate input/output structures, inefficient memory utilization, and poorly defined timing. Subroutines written using these high-level languages cannot be used in a PDCP software system which relies upon software timing unless the resulting machine code is investigated to determine all possible execution times. Even for an interrupt-timed system, the maximum execution time of the subroutines must be determined to insure that processing can be completed between transmissions.

4.3 PDCP SOFTWARE DEVELOPMENT

As presented above, neither assembly languages nor standard high-level languages are ideally suited for the development of PDCP software systems. This problem is not unique to the data collection field but is of general concern in the areas of mini- and microcomputer programming. Therefore, a new program development technique which retains the advantages of both high-level and assembly languages should be developed.

4.3.1 Applications-Oriented Software Development

A simplified technique for microcomputer programming by applications-oriented nonprogrammers has recently been proposed by Korn [1]. This technique is based on a software system organization nearly identical to the one illustrated in Figure 4.1(1). Korn [1] assumes that a set of well-defined assembly language, block-operator subroutines are available to perform each of the subfunctions which may be required by an applications program. The applications program is specified by means of a block-diagram employing the standard functions. An interactive editor/translator program running on a small minicomputer is used to translate the block-diagram specifications into an address table. The address table serves the same function as the executive program proposed in Figure 4.1(1). During program execution, the address table specifies successive jumps to the standard subroutines required to correctly execute

the applications program. A choice between the address table and executive program techniques will depend upon the extent to which indirect addressing and subroutine calls are supported by a particular microprocessor's instruction set.

Korn [1] states that the editor/translator programming system should satisfy the following requirements:

1. The language used to communicate with the editor/translator program should be easy to learn and understand.

2. Simple elementary operations well known to engineers and scientists should be used to build up complex operations.

3. The language should be entirely independent of the specific microcomputer used.

4. The editor/translator system should generate microprocessor code as efficient in the use of time and memory as that developed by a good assembly language programmer.

These goals are applicable to the proposed PDCP program development system but are not sufficient. A PDCP program development system should also satisfy the following requirements:

1. In addition to the elementary operations, as many major PDCP system programs as possible should be available to the user in block form. For example, this would include transmitter programs (see Section 3.3) and the zero-order floating aperture predictor subroutine (see Section 3.2).

2. Memory requirements should be indicated to the user in a form that is representative of memory cost. This implies that memory required for storage of the executive program (usually, user defined ROM or PROM) should be listed separately from data storage or scratch pad RAM memory. Also, the number of firmware ROM modules required for storage of the standard subroutines should be indicated.

3. Program execution time should be indicated to the user, and an error should be noted if the program execution time exceeds a specified transmission interval.

4. Alternate subroutine structures should be available for software and interrupt-timed systems. This is necessary to obtain maximum system throughput with minimum program storage area.

A PDCP software development system with the capabilities described above could significantly enhance usage of PDCP's by applications-oriented nonprogrammers.

4.3.2 Assembly Language and Microprogrammed Subroutines

To insure efficiency and accurate timing, the functional block subroutines employed in the programming system described above should be developed by experienced programmers using assembly language and/or microprogramming techniques. Within the constraints of a given instruction set, assembly language programming can produce the optimum realization of a particular software task. Many microprocessors have general purpose instruction sets which are fixed by hardwired control logic integrated within the microprocessor chip itself. These instruction sets must be individually evaluated to determine their relative merits in particular applications such as PDCP programming (see Section 6). In contrast, the instruction set of a microprogrammable microprocessor can be optimized for a particular application. A microprogrammed microprocessor utilizes a programmed logic control unit rather than a hardwired control unit. This permits the user to define and sequence microprocessor operations at the most elementary level. As a result, the power of a microprogrammed instruction set is limited only by the amount of control storage available and the capabilities of the microprocessor's arithmetic/logic unit. In fact, when sufficient control storage is available, microprogramming need not be limited to the development of a simple instruction set. Those operations which are used most frequently, or require minimum execution time, can be implemented as microprogrammed subroutines. Regardless of whether or not the microprocessor chosen for the PDCP has a microprogrammable instruction set, PDCP operations which are not included in the

machine's basic instruction set can be implemented using assembly language programming. Examples of assembly language subroutines which could be included in the library of the editor/translator program are presented in the following section.

4.4 PDCP PROGRAM EXAMPLES

The hardwired logic control units used in most current DCP designs perform four basic tasks: data input, elementary sensor control, data formatting, and transmitter control. In this section, example programs are presented to demonstrate the ability of a microprocessor based control unit to perform each of these tasks. These example programs show that, even when a microprocessor based control unit is used to perform all basic DCP tasks, excess computational capability exists. This capability can be used for data preprocessing or special computations which would reduce the load on the satellite data link. Fourier analysis of seismic data is presented as a potential special computation. Examples of data preprocessing subroutines are also shown.

Unless otherwise indicated, the programs listed in the following sections have been written to be executed on the 8080A-based PDCP demonstration system (see Section 5). Standard Intel Corporation 8080A mnemonics [2] have been used except within macroinstructions. Instructions such as NOP and MOV A,A are sometimes used to produce a required delay without otherwise affecting program execution. Macro delay instructions have been defined so that this usage of an instruction will be clear within the context of the program. The mnemonic DLY XX is used for these macroinstruction groups. The decimal number which replaces XX indicates the length of the delay in states. Since the assembly language program listings were produced using a PDP 11/40 minicomputer, constants and addresses are represented as octal numbers except as noted by the use of a decimal point.

4.4.1 Data Input and Sensor Control

Data input and elementary sensor control are two of the four basic tasks performed by the hardwired control logic currently used in most

DCP designs. A microprocessor based control unit is ideally suited to performing these tasks. Each PDCP sensor can be serviced by a simple subroutine which provides the functions of data input and sensor control. Because of the versatility of the microprocessor, the cost and complexity of the data acquisition channels can potentially be reduced without a significant increase in control logic. Subroutines which demonstrate microprocessor control of the data input and sensor control functions are listed below.

4.4.1.1 Frequency Measurement - Many sensors currently in use on DCP's produce an output signal whose frequency is proportional to the parameter of interest (temperature or air pressure for example). Program 4.4.1.1(1) has been developed to demonstrate that a PDCP can digitize the output of these sensors directly. A flow chart for Program 4.4.1.1(1) is illustrated in Figure 4.4.1.1(1). This frequency counter program is presented in the form of a subroutine which is suitable for use as an input block operation in the programming system described in Section 4.3.1. The sensor output signal is input to the microprocessor on one bit of input port zero. Signal frequency is determined by counting 0 to 1 logic level transitions for a preset interval of time. Two parameters, the number of samples to be taken and the input bit assigned to the sensor, are passed to the subroutine from the calling program. The input bit assignment for the sensor is specified by a mask byte in register C. This byte contains a one in the bit position corresponding to the bit position of the input signal and zero's in all other bits. Thus, a single frequency counter subroutine can service up to eight sensors without multiplexing. The number of samples to be taken is specified by the contents of register pair DE. Since the number of samples is proportional to the measurement interval, the resolution and scale factor of the digitization process are determined by the number of samples taken. With a measurement interval of one second, the sensor frequency is measured directly in units of Hertz (Hz). The binary output is returned to the calling program in register pair DE.

The sampling interval establishes an upper bound on the frequency which can be measured using this subroutine. A minimum sampling interval

```
;                    FREQUENCY SENSOR INPUT SUBROUTINE                    ;
;                                                                         ;
;  THIS SUBROUTINE IS USED TO CONVERT A FREQUENCY SENSOR OUTPUT TO        ;
;  PARALLEL DATA.   THE NUMBER OF SAMPLES TO BE TAKEN IS PRELOADED        ;
;    BY THE EXECUTIVE IN THE (DE) REGISTER PAIR.  SINCE A FIXED           ;
;    AMOUNT OF TIME IS REQUIRED TO TAKE A SAMPLE, THE NUMBER OF           ;
;    SAMPLES TAKEN IS PROPORTIONAL TO THE SAMPLING PERIOD.   77           ;
;    MACHINE STATES ARE REQUIRED TO MAKE ONE SAMPLE.  THE VALUE           ;
;    OF THE COUNT IS RETURNED IN THE (DE) REGISTER PAIR.  UP TO           ;
;    EIGHT DIFFERENT INPUT CHANNELS CAN BE SAMPLED,    ONE AT A          ;
;    TIME.  THE EXECUTIVE MUST PLACE THE BIT MASK IN THE A REG-           ;
;    PRIOR TO CALLING THE COUNT ROUTINE.    THE MASK DETERMINES           ;
;    WHICH BIT OF INPUT PORT 0 IS TO BE SAMPLED.  CPU STATUS OF           ;
;    B,C,H,L REGISTERS PRESERVED.                                         ;
;                                                                         ;
; 86 + 77*#SAMPLES TAKEN MACHINE STATES, 36 BYTES ROM AND 6 BYTES         ;
;                              STACK.                                     ;

000000                          COUNT:
000000                                  PUSH    B               ; SAVE CPU STATUS OF BC AND HL
000000  305
000001                                  PUSH    H               ; REGISTER PAIRS
000001  345
000002                                  LXI     H,0             ; ZERO COUNT REGISTERS
000002  041     000     000
000005                                  MVI     C,1             ; 1 ----> C
000005  016     001
000007                                  MOV     B,A             ; PUT BIT MASK PASSED FROM EXECUTIVE INTO B
000007  107
000010                          C0:
000010                                  IN      0               ; INPUT SAMPLE OF SIGNAL
000010  333     000
000012                                  ANA     B               ; PICK OFF APPROPRIATE BIT BY MASKING WITH B
000012  240
000013                                  JZ      C2              ; NOT POSITIVE EDGE, DON'T INCREMENT COUNT
000013  312     037     000
000016                                  CMP     C               ; COMPARE TO LAST SAMPLE
000016  271
000017                                  JZ      C3              ; NOT A TRANSITION SO DON'T INCREMENT COUNT
000017  312     042     000
000022                                  INX     H               ; IT'S A TRANSITION AND A POS. EDGE, SO INCREMENT COUNT
000022  043
000023                                  DLY5                    ; DELAY FOR EQUAL TIME BRANCHES
000023  177
```

Program 4.4.1.1(1) Frequency Sensor Input Subroutine.

```
000024                          C1:
000024                                  MOV     C,A             ; STORE SAMPLE IN C
000024  117
000025                                  DCX     D               ; DECREMENT TIME COUNT
000025  033
000026                                  MOV     A,E             ; CHECK FOR ZERO IN DE
000026  173
000027                                  ORA     D
000027  262
000030                                  JNZ     C0              ; NOT ZERO TIME OUT SO SAMPLE AGAIN
000030  302     010     000
000033                                  XCHG                    ; PUT COUNT FROM HL INTO DE
000033  353
000034                                  POP     H               ; RESTORE CPU STATUS
000034  341
000035                                  POP     B
000035  301
000036                                  RET                     ; RETURN TO EXECUTIVE
000036  311

000037                          C2:     DLY14                   ; ADD DELAY SO THREE PATHS ARE
000037  177     177
000041  000
000042                          C3:     JMP     C1              ; EQUAL TIME WISE
000042  303     024     000

000045                                  END
```

Program 4.4.1.1(1) (continued)

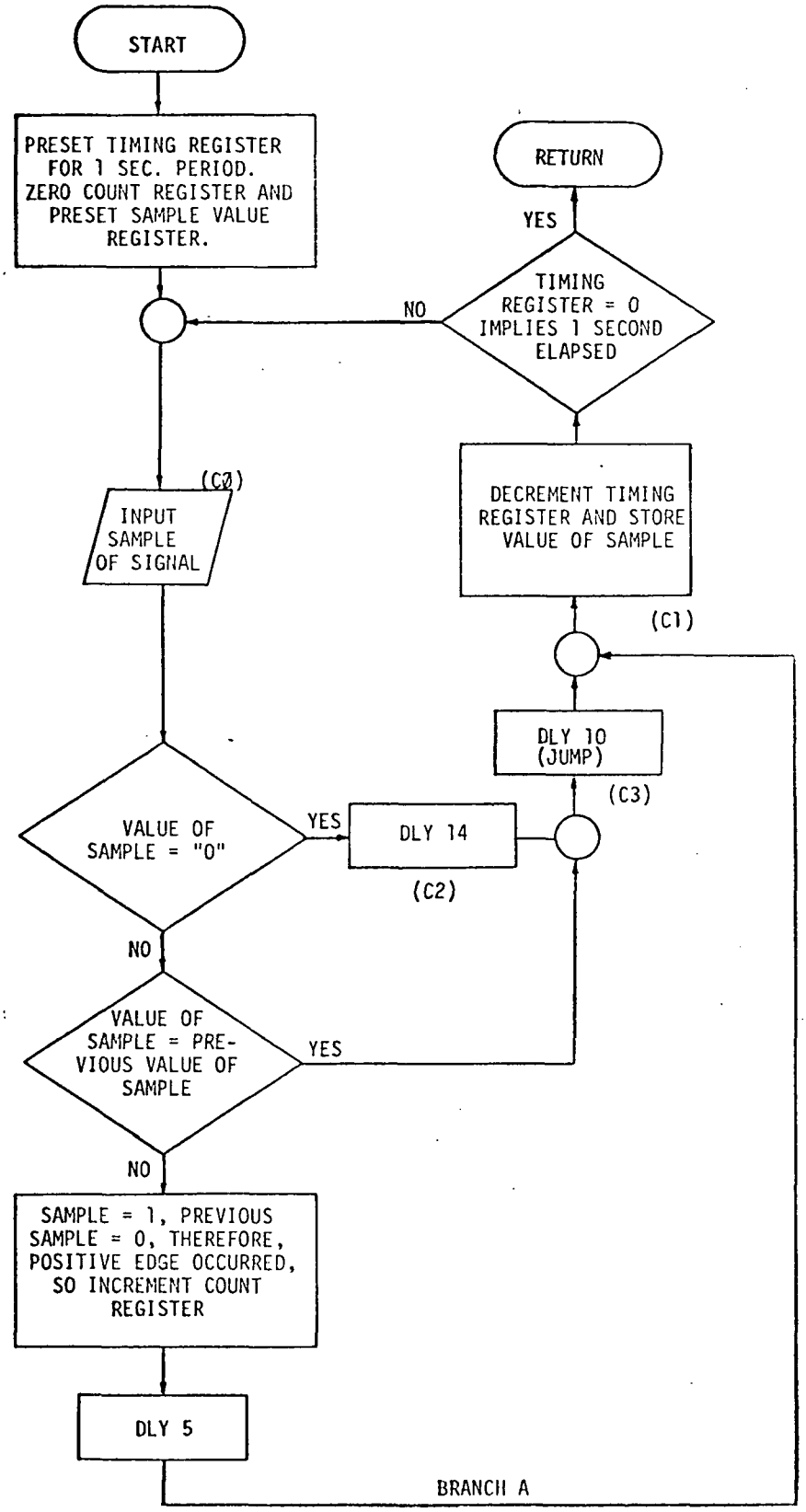


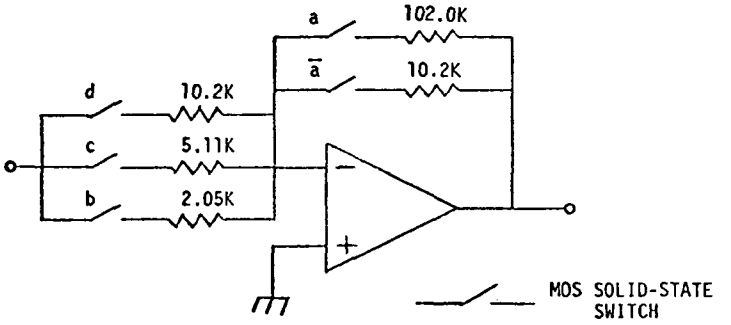
Figure 4.4.1.1(1) Flow Chart for Frequency Sensor Input Subroutine.

is desirable, but the interval must be constant. As shown in Figure 4.4.1.1(1), the program executes one of three branches depending upon the value of the current and the preceding sample. The longest natural program branch (Branch A) occurs when a positive signal transition is detected. Delay must be added to the other two program loops so that all three loops will execute in the same number of states. Unfortunately, arbitrary delays are not possible because the microprocessor is synchronized to a constant frequency clock. As a result, a minimum delay of five states must be added to Branch A so that the execution time of all three branches can be equalized.

Program 4.4.1.1(1) requires 77 states between samples of the input signal. Each state is a minimum of 500 nsec for an 8080A microprocessor or 325 nsec for a high speed 8080A-1 microprocessor. Thus, the maximum sampling frequencies are 25,974 Hz and 39,960 Hz, respectively. From the sampling theorem [3], these sampling frequencies establish upper bounds on the input signal frequencies of 12,987 Hz and 19,980 Hz, respectively. Frequency prescaling will be required for sensors which produce higher output frequencies.

4.4.1.2 Autoranging - In some PDCP applications, the parameter of interest is a relatively slowly changing function of time which can vary over a wide range. The design of an accurate data acquisition channel for parameters of this type is difficult because linearity must be maintained over a wide dynamic range. If the data acquisition channel gain is set to produce a detectable signal level for the minimum sensor output, the later stages of the data acquisition channel may saturate as the sensor output increases. A solution to this problem is to vary the gain of the data acquisition channel so that the signal is always within the most linear and accurate system dynamic range. This can be accomplished by including a logarithmic amplifier in the front end of the data channel. The logarithmic amplifier, however, introduces a log conformity error which is unsatisfactory for systems requiring high resolution throughout the voltage range. In addition, the logarithmic scaling of the digitized data is inconvenient for most PDCP data preprocessing subroutines. These problems can be avoided by using a variable gain amplifier in place of the logarithmic amplifier.

The concept of a microprocessor-controlled variable gain amplifier is illustrated in Figure 4.4.1.2(1). A block-diagram of a typical data acquisition system using the variable gain amplifier is shown in Figure 4.4.1.2(2). Channel 1 is designed to acquire data from a sensor which produces a slowly varying, wide-range (200:1) output voltage. This signal is digitized by the analog-to-digital converter and input to the PDCP's microprocessor. Program 4.4.1.2(1) controls data acquisition on Channel 1. A flow chart for Program 4.4.1.2(1) is shown in Figure 4.4.1.2(3). Each time the executive PDCP program calls the data acquisition subroutine for Channel 1, a sample is obtained. If the sample is between 10 percent and 90 percent of full scale, the value of the sample and a code representing the current channel gain are stored in memory for later transmission. Whenever the sample is not within the ideal range of 10 percent - 90 percent of full scale, the gain of the amplifier is adjusted and another sample is obtained. Thus, a maximum system accuracy and an eight-bit resolution are maintained over a 200:1 dynamic range.



NOMINAL GAIN	CONTROL SIGNALS abcd
-1	0001
-2	0010
-3	0011
-5	0100
-7	0110
-10	1001
-20	1010
-30	1011
-50	1100
-70	1110

Figure 4.4.1.2(1) Microprocessor-Controlled Variable-Gain Amplifier.

```
                        ;         AMPLIFIER VARIABLE GAIN CONTROL ROUTINE          ;
                        ;                                                          ;
                        ;  THIS ROUTINE IS USED TO VARY THE GAIN OF A VARIABLE     ;
                        ; GAIN AMPLIFIER.  THE AMPLIFIER GAIN IS VARIED BETWEEN   ;
                        ; -1.0 AND -70.0 TO KEEP THE ADC INPUT VOLTAGE BETWEEN    ;
                        ;  10% AND 90% OF FULL SCALE   (5.0 VOLTS)                ;

000000                  VGC:                            ; START OF VAR GAIN CONTROL ROUTINE
000000                          PUSH    PSW             ; SAVE WORKING REGISTERS SO
000000  365
000001                          PUSH    B               ; THAT THE MAIN PROGRAM IS
000001  305
000002                          PUSH    D               ; NOT ALTERED
000002  325
000003                          PUSH    H
000003  345
000004                          LHLD    2000            ; PLACE ADDRESS OF CONTROL WORD
000004  052     000     004
                                                        ; INTO REGISTERS H AND L
000007                  SAMPLE: MOV     A,M             ; PLACE CONTROL WORD IN REGISTER A
000007  176
000010                          OUT     3               ; LATCH CONTROL WORD INTO OUTPUT PORT 3
000010  323     003
000012                          STA     2002            ; STORE CODE FOR GAIN SETTING IN MEMORY
000012  062     002     004
000015                          LXI     B,15346         ; LOAD BYTE REPRESENTING
000015  001     346     032
                                                        ; 0.51 V INTO REGISTER B AND BYTE
                                                        ; REPRESENTING 4.49 V INTO REGISTER C
                                                        ; WHILE WAITING FOR ADC RESULT
000020                          IN      3               ; LOAD A/D CONVERTER RESULT INTO REGISTER A
000020  333     003
000022                          CMP     C               ; CHECK FOR OVERRANGE
000022  271
000023                          JNC     DEC             ; TRY TO DECREASE GAIN IF OVERRANGE
000023  322     045     000
000026                          CMP     B               ; CHECK FOR UNDERRANGE
000026  270
000027                          JC      INC             ; TRY TO INCREASE GAIN IF UNDERRANGE
000027  332     057     000
000032                  STORE:  STA     2003            ; STORE CORRECT ADC DATA IN MEMORY
000032  062     003     004
                                                        ; FOR LATER PROCESSING BY MAIN PROGRAM
                                                        ; OR TO AWAIT TRANSMISSION
000035                          SHLD    2000            ; STORE LAST CONTROL WORD ADDRESS
000035  042     000     004
000040                          POP     H               ; RESTORE WORKING REGISTERS AND
000040  341
000041                          POP     D               ; RETURN TO MAIN PROGRAM
000041  321
000042                          POP     B
000042  301
000043                          POP     PSW
000043  361
000044                          RET
```

Program 4.4.1.2(1) Amplifier Variable Gain Control Subroutine.

```
000044  311
000045                      DEC:    MVI     D,31
000045  026     031
000047                              CMP     D               ; CHECK FOR GAIN=-1
000047  272
000050                              JZ      STORE           ; STORE PREVIOUSLY SAMPLED DATA IF TRUE
000050  312     032     000
000053                              DCR     L               ; IF NOT, DECREASE GAIN AND SAMPLE
000053  055
000054                              JMP     SAMPLE          ; AGAIN
000054  303     007     000
000057                      INC:    MVI     D,351
000057  026     351
000061                              CMP     D               ; CHECK FOR GAIN=-70
000061  272
000062                              JZ      STORE           ; STORE PREVIOUSLY SAMPLED DATA IF TRUE
000062  312     032     000
000065                              INR     L               ; IF NOT, INCREASE GAIN AND SAMPLE
000065  054
000066                              JMP     SAMPLE          ; AGAIN
000066  303     007     000
```

Program 4.4.1.2(1) (continued)

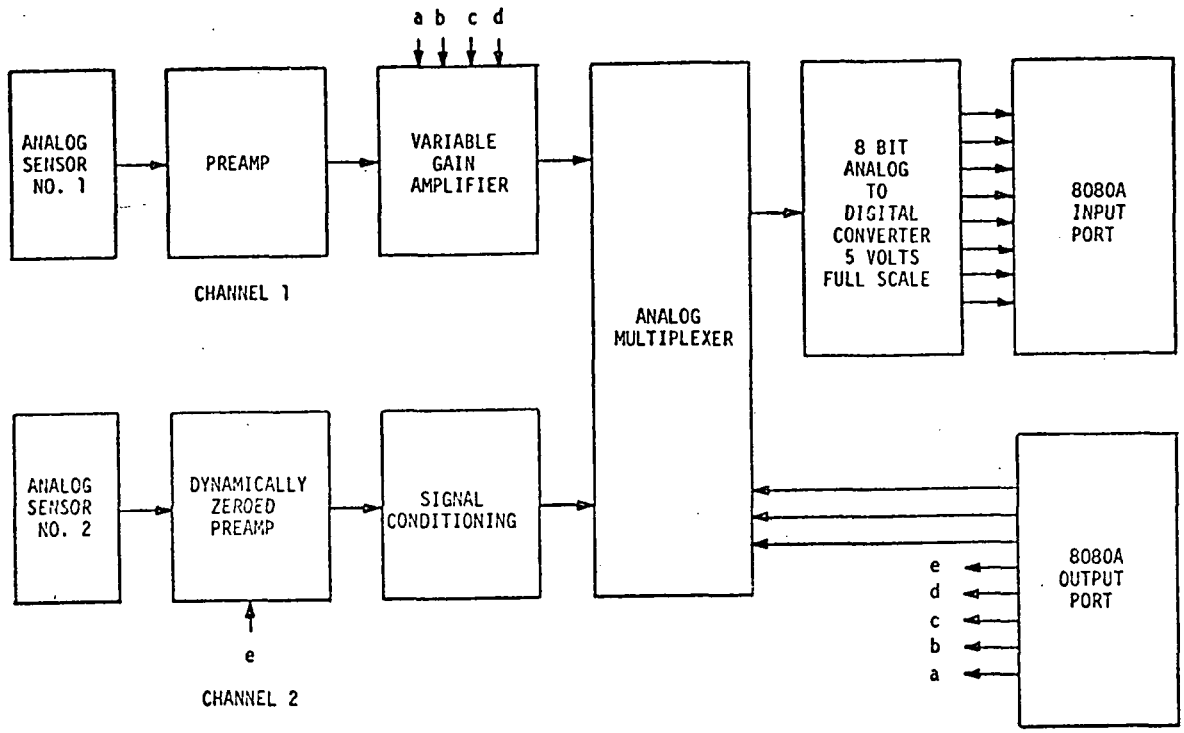


Figure 4.4.1.2(2) Analog Data Acquisition System for a PDCP.

4.4.1.3 Software Controlled Analog-to-Digital Conversion - The traditional analog-to-digital converter (ADC) consists of a digital-to-analog converter, a comparator, and some control logic. A digital word loaded into the digital-to-analog converter (DAC) produces an analog output signal (voltage or current) which is applied to one input of the comparator. The second comparator input is the analog signal which is to be digitized. The comparator output signal indicates whether the analog input is greater than or less than the reference signal produced by the DAC. This signal is fed back to the control logic which uses a fixed algorithm to update the DAC input. The above process continues until the DAC output equals the analog signal input. At that point, the digital word in the DAC is the digital representation of the analog input voltage.

A number of different algorithms can be employed in the analog-to-digital conversion process but one of the most widely used is successive approximation. In a successive approximation ADC, the unknown analog

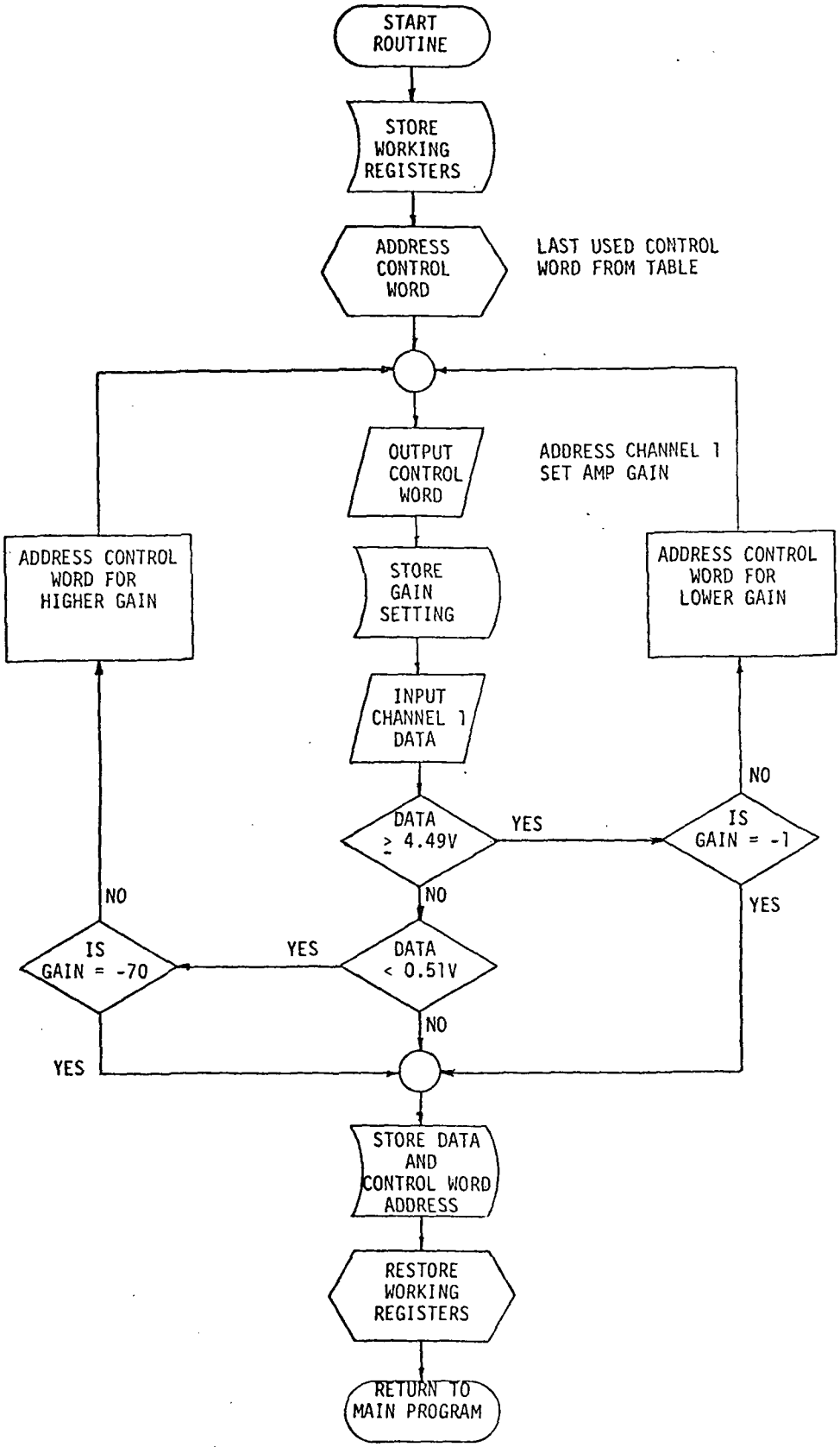


Figure 4.4.1.2(3) Flow Chart for Variable Gain Amplifier Control Program.

input is compared to the DAC output with only the most significant bit (MSB) true. This DAC output corresponds to one-half of full scale. If the input is greater than the MSB, the MSB is left in the logical "1" or true state and the next bit (which corresponds to one-quarter of full scale) is set to a logical "1". If the second bit does not cause the DAC output to exceed the unknown analog input, it is left in the logical "1" state and the next bit is set to "1". If the second bit causes the DAC output to exceed the unknown analog signal, it is set back to a logical "0" as the next bit is set to a logical "1" for another comparison. This process continues in order of decending bit weight until the proper state of the last bit has been determined. At that time the digital input to the DAC represents the correct digitization of the previously unknown analog voltage or current.

All of the digital control logic normally found in the successive approximation ADC can be replaced by a software subroutine using the microprocessor. Program 4.4.1.3(1) is a successive approximation analog-to-digital conversion subroutine written for the UT PDCP development system. A flow chart for this program is illustrated in Figure 4.4.1.3(2) and a simplified schematic of the ADC hardware used in the development system is shown in Figure 4.4.1.3(3).

The successive approximation subroutine uses only 34 bytes of ROM and six bytes of RAM. As currently implemented, the analog-to-digital conversion requires a fixed time of 748 states. For a standard 8080A operating with a 2-MHz clock, this corresponds to 2673 conversions per second, a rate which should be more than satisfactory for the PDCP application.

4.4.2 Data Compaction and Preprocessing

The amount of data which must be handled by data collection satellites is constantly increasing. The major factors contributing to this increase are the use of high sampling-rate sensors and a general growth in the number of DCP's. PDCP's can be expected to contribute substantially to the increase in remote data collection activities by opening new areas of application. In addition to taxing the bandwidth limits of

```
                                ;                SOFTWARE ANALOG DIGITAL CONVERSION                 ;
                                ;                                                                   ;
                                ; THIS ROUTINE SAVES ALL MACHINE STATUS EXCEPT THE ACCUMULATOR      ;
                                ; AND FLAGS.  THE RESULTING ANALOG TO DIGITAL CONVERSION IS         ;
                                ; RETURNED IN THE ACCUMULATOR.  THE TECHNIQUE EMPLOYED IS THE       ;
                                ; SUCCESSIVE APPROXIMATION ALGORITHM.                               ;
                                ;                                                                   ;
                                ;  ROUTINE REQUIRES 34 BYTES OF ROM, 6 BYTES OF STACK AND 748      ;
                                ;  MACHINE STATES.  THIS IS A CONSTANT EXECUTION TIME ROUTINE       ;
                                ;  WHICH COULD BE SHORTENED.                                        ;
000000                          ADC:
000000                                  PUSH    B               ; SAVE MACHINE STATUS
000000  305
000001                                  PUSH    D
000001  325
000002                                  LXI     B,1             ; 1 ----> C, 0 ----> B
000002  001     001     000
000005                                  MVI     D,8.            ; 8. ----> D
000005  026     010
000007                          ADC0:   MOV     A,C             ; GET OLD PIVOT
000007  171
000010                                  RRC                     ; ROTATES TO NEXT LOCATION
000010  017
000011                                  MOV     C,A             ; SAVE IT IN C
000011  117
000012                                  ORA     B               ; CREATE TEST BYTE
000012  260
000013                                  OUT     DAC             ; OUTPUT TO DAC PORT
000013  323     004
000015                                  IN      DACSTA          ; INPUT DAC COMPARATOR STATUS
000015  333     003
000017                                  RRC                     ; ROTATE COMPARATOR OUTPUT TO CARRY
000017  017
000020                                  JNC     ADC2            ; ANALOG VOLTAGE NOT GREATER THAN TEST, DON'T KEEP IT
000020  322     036     000
000023                                  MOV     A,C             ; ELSE KEEP BIT
000023  171
000024                                  ORA     B               ; CREATE BIT OF PARTIAL CONVERSION
000024  260
000025                                  MOV     B,A             ; SAVE PARTIAL CONVERSION
000025  107
000026                          ADC1:   DCR     D               ; CHECK FOR LAST BIT TESTED
000026  025
000027                                  JNZ     ADC0            ; NOT LAST, TEST ANOTHER
000027  302     007     000
000032                                  MOV     A,B             ; PUT RESULT IN ACCUMULATOR
000032  170
000033                                  POP     D               ; RESTORE MACHINE STATUS
000033  321
000034                                  POP     B
000034  301
000035                                  RET                     ; ELSE RETURN TO CALLER
000035  311
000036                          ADC2:   DLY4                    ; DELAY FOR EQUAL TIME LOOPS
000036  000
000037                                  JMP     ADC1            ; JUMP TO BIT NUMBER TEST AND CONTINUE
000037  303     026     000
```

Program 4.4.1.3(1) Successive Approximation Analog-to-Digital Conversion Subroutine.

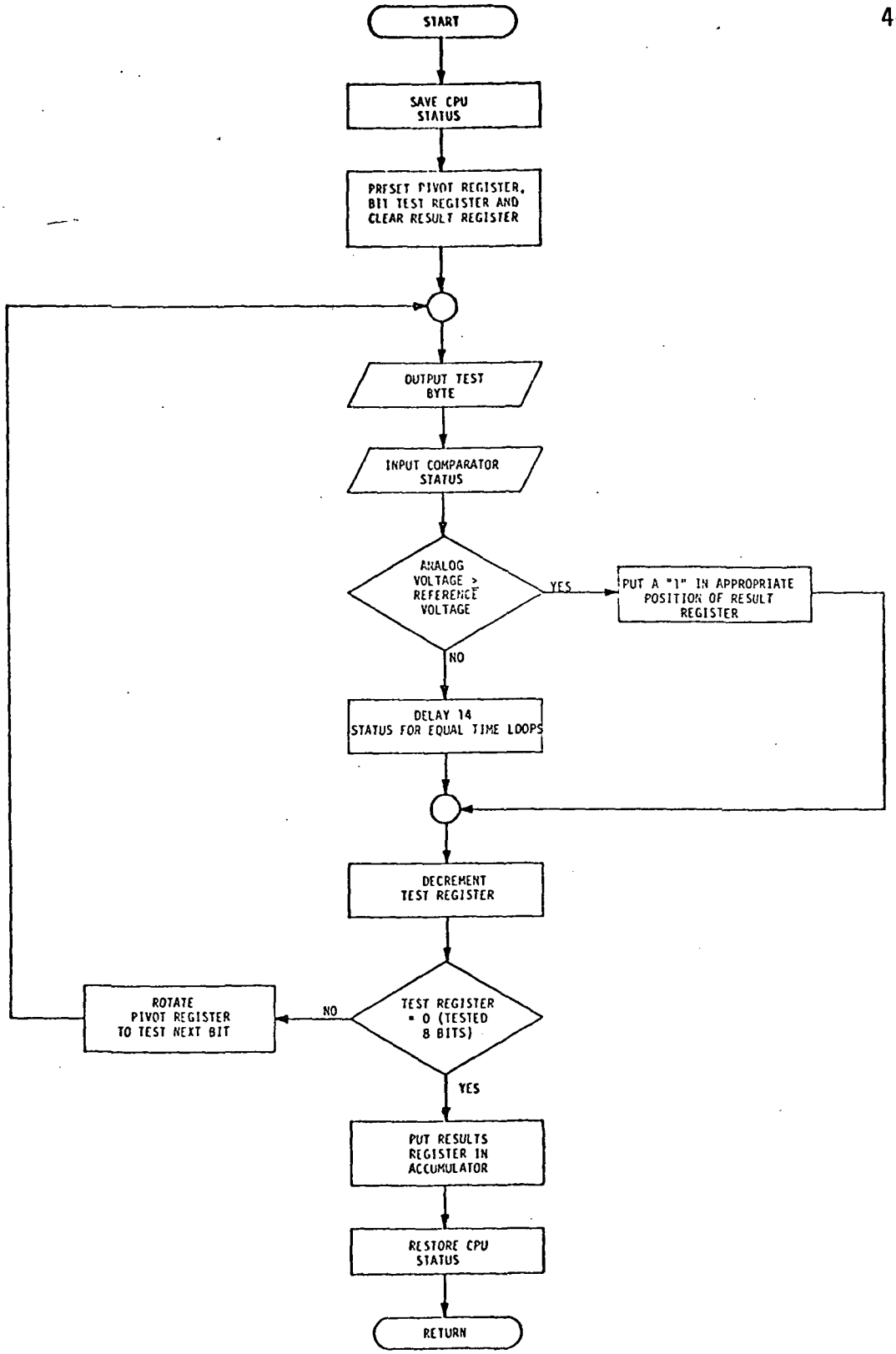


Figure 4.4.1.3(1) Flow Chart for Successive Approximation ADC Subroutine.

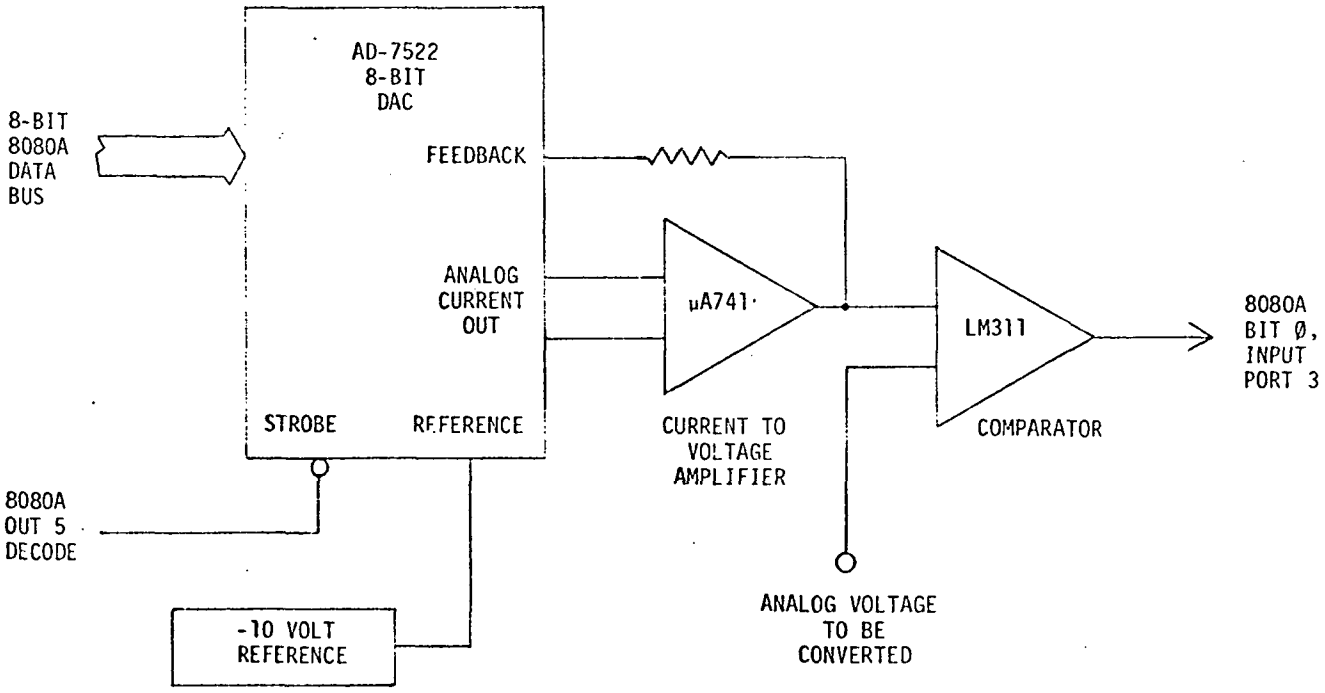


Figure 4.4.1.3(2) Simplified Schematic of the ADC Hardware.

present data collection satellites, significant increases in data flow would require corresponding increases in the capacity of present ground-to-ground communication lines, data storage areas, and data processing facilities. Therefore, techniques should be developed for reducing the volume of data flow without significantly reducing the amount of information acquired.

One of the most useful ways to reduce data flow to manageable proportions is to apply compression or compaction before the data is transmitted from the PDCP. There are two basic types of data compression techniques: entropy-reducing and information-preserving [4]. Entropy-reducing algorithms perform an irreversible transformation on the input data and therefore cause a pre-transmission loss of some information. Information-preserving algorithms perform a reversible transformation on the input data. As a result, the information carried by the original data can be recovered within a specified allowable tolerance or peak error. Bit compaction rates of two-to-one can normally be achieved by information-preserving algorithms. Much higher degrees of data compaction can be obtained by using entropy-reducing algorithms.

Data compaction techniques have not been employed in previous DCP designs because of the extra hardware and expense involved in hardwired implementations of the algorithms. Many data compression algorithms, however, can be economically used with PDCP's. In most cases, the only additional hardware required to provide data compression capabilities will be the memory used to store subroutine implementations of the various data compression algorithms. Data compression subroutines are excellent candidates for the subroutine library within the applications-oriented PDCP programming system (see Section 4.3.1). Several examples of data compression programs are presented in the following sections.

4.4.2.1 General-Purpose Binary Math Package - Most data compression algorithms are based upon mathematical operations. Therefore, a general-purpose binary math package has been developed for use in data compaction operations. The math package consists of subroutines which perform the unsigned binary operations specified in Table 4.4.2.1(1). Unsigned binary format is assumed since PDCP data will usually be represented in this

TABLE 4.4.2.1(1)

GENERAL PURPOSE PDCP BINARY MATH PACKAGE

Basic Operation	Form (Allowable Number Size In Bytes)	Memory Requirements In Bytes		Execution Time In States (N=Number of Bytes)	Subroutine Title
		Program Storage (ROM)	Stack (RAM)		
ADDITION	Double Precision (2+2→2)	11	2	83	DADDM
	Short Multiprecision (2+4→4)	20	4	141	SMADD
	Multiprecision ([1-256]+[1-256]→[1-256])	17	10	56N+95	MADD
SUBTRACTION	Double Precision (2+2→2)	11	2	83	DSUBM
	Short Multiprecision (2+4→4)	20	4	141	SMSUB
	Multiprecision ([1-256]+[1-256]→[1-256])	17	10	56N+95	MSUB
MULTIPLICATION	Double Precision (2x2→4)	73	14	4683-4843	DPMULT
DIVISION	Double Precision (4÷2→2)	64	8	3645-8456	DPDIV
SQUARE ROOT	Double Precision ($\sqrt{2}$ →1)	84	14	4336-4746	SQRT

TABLE 4.4.2.1(1) (Continued)

Basic Operation	(Allowable Number Size In Bytes)	Memory Requirements In Bytes		Execution Time In States (N=Number of Bytes)	Subroutine Title
		Program Storage (ROM)	Stack (RAM)		
COMPARISON	Double Precision (2-2)	12	4	62-73	DCMPM
	Multiprecision ([1-256]-[1-256])	54	10	110N+108	MCMP
MEMORY-TO-MEMORY MOVE	Multiprecision ([1-256]MOVE TO[1-256])	15	10	49N+91	MOVE
SUBROUTINE EXIT RESTORING ALL STATUS	POPS ALL STATUS	5	-	60	EXIT

form. Listings of the math package subroutines are provided as Programs A(1) through A(12) in Appendix A.

The most important PDCP math subroutines (addition and subtraction) are available in three formats: double precision (numbers up to 16 bits long), short multiprecision (mixed 16-bit and 32-bit number operations), and multiprecision (numbers up to 2048 bits long). This allows the PDCP programmer to consider memory requirements, execution time, and operand resolution in selecting the best subroutine for a particular task. Most PDCP data words will be eight to ten bits long and can most appropriately be manipulated using the double precision subroutines. The short multiprecision subroutines supplement the double precision subroutines and are intended for use in operations such as summing a large number of data points where the result could overflow 16 bits (i.e., a result greater than 65,535). Short multiprecision and double precision subroutines should be sufficient for PDCP data compaction operations. Multiprecision subroutines capable of operating on numbers up to 256 bytes long are provided primarily to illustrate that numbers of any magnitude can be processed by a microprocessor based PDCP if sufficient calculation time is available.

The comparison, multiplication, division, and square-root subroutines permit significant data compaction operations such as data averaging to be performed by the PDCP. Since a special square-root algorithm developed by Kostopoulos [5] was implemented, a flow chart for the square-root subroutine is shown in Figure 4.4.2.1(1). The MOVE subroutine is provided for transferring multiprecision data blocks between temporary storage and working memory areas when using multiprecision arithmetic subroutines. EXIT is a general purpose block of code which is used to restore all CPU status when returning from selected subroutines. The complete binary math package is 403 bytes long and would occupy only 40 percent of a 8K ROM.

Normally, a PDCP will not require all of the subroutines listed in Table 4.4.2.1(1). A basic math package consisting of double and short multiprecision addition and subtraction, double precision compare, and EXIT would be sufficient for elementary data compaction operations. This

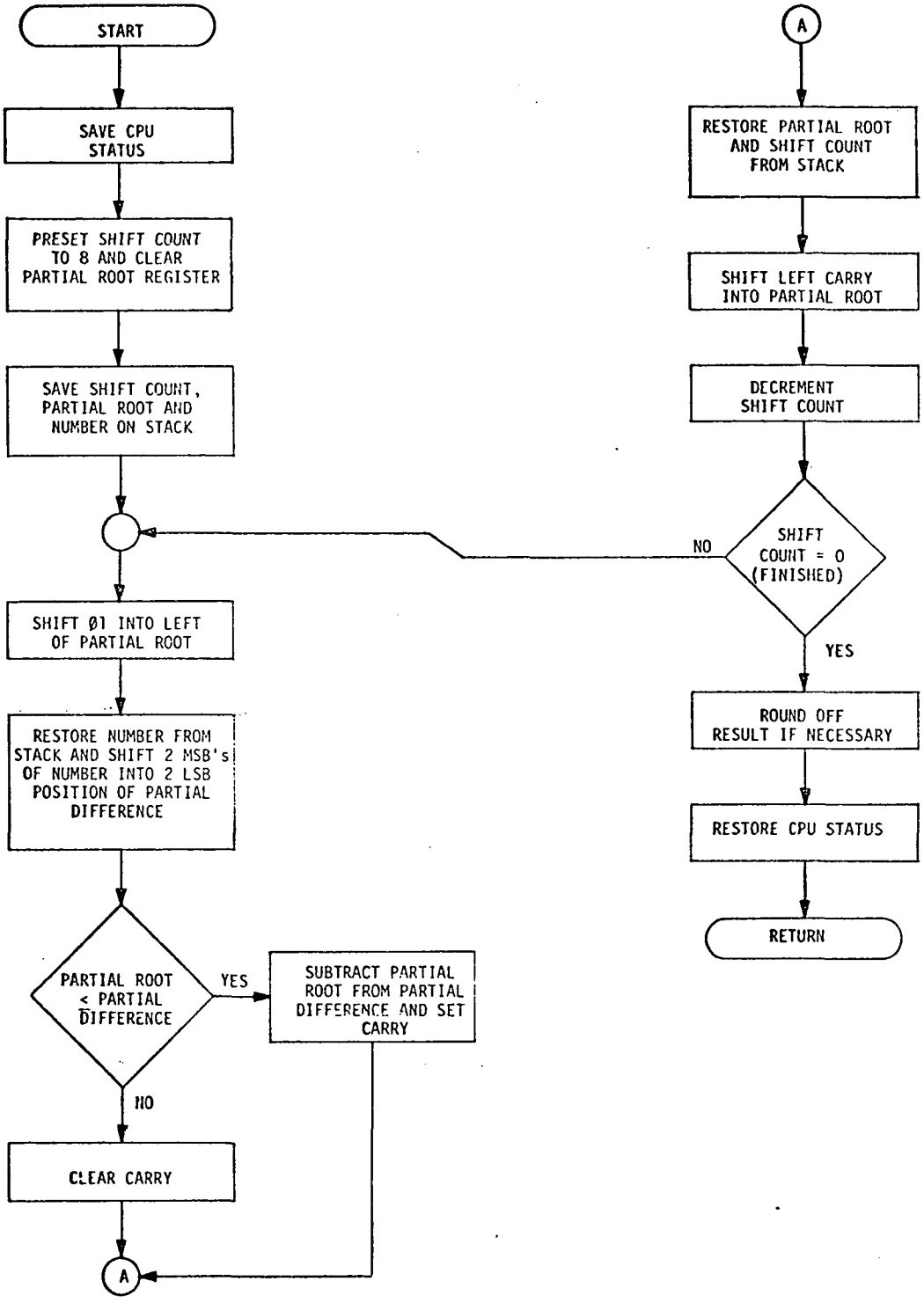


Figure 4.4.2.1(1) Flow Chart for Square Root Subroutine.

basic math package can be stored in 79 bytes of ROM. Multiplication and division can be added to the basic math package to support more advanced data compaction operations such as data averaging. A total of 216 bytes are required to store this form of the math package. Standard deviation calculations require the addition of a square-root subroutine which increases the memory storage area used by the extended version of the basic math package to 300 bytes. Execution times of the PDCP binary math subroutines are shown in Table 4.4.2.1(2).

4.4.2.2 Determination of Minimum and Maximum Data Values - Program 4.4.2.2(1) is an entropy-reducing data compaction subroutine which determines the minimum and maximum data samples obtained from a sensor over a particular time interval. The DMINMAX program uses the double precision compare subroutine from the binary math package and operates on numbers up to 16 bits long. Each time the DMINMAX routine is called by the PDCP executive program, the value of a new data point is compared to previously established upper and lower bounds. The carry and zero flags indicate the result of this comparison. If the new data point is outside one of the boundaries, the data point is stored as a new boundary. Therefore, just prior to transmission, the upper and lower bounds will represent the minimum and maximum data samples obtained since the last transmission. Program 4.4.2.2(1) is 29 bytes long and requires eight bytes of RAM for stack operations. Worse case execution time is 264 states which would permit a data rate of over 7,500 samples per second.

A multiprecision MINMAX subroutine is listed as Program 4.4.2.2(2). The MINMAX program uses two subroutines from the multiprecision binary math package and operates on numbers up to 256 bytes long. The value of a memory location named TMCODE indicates the result of the MINMAX program execution. Program 4.4.2.2(2) is 120 bytes long and requires 20 bytes of RAM for stack operations. These figures include the required multibyte COMPARE and MEMORY-TO-MEMORY MOVE subroutines from the math package.

4.4.2.3 Zero-Order Floating Aperture Predictor - A potentially more useful data compression algorithm is implemented in Program 4.4.2.3(1). The zero-order floating aperture predictor algorithm is an information-preserving polynomial data compression technique [4]. The algorithm is

TABLE 4.4.2.1(2)

BINARY MATH SUBROUTINE EXECUTION TIMES

Operation	Number Size (8-bit Bytes)	Execution Time		
		States	Microseconds Using 500-nsec 8080A	Microseconds Using 325-nsec 8080A-1
ADD	2	83	41.5	26.98
	2,4	141	70.5	45.83
	N	95+56N	47.5+28N	30.88+18.2N
SUBTRACT	2	83	41.5	26.98
	2,4	141	70.5	45.83
	N	95+56N	47.5+28N	30.88+18.2N
COMPARE	2	62-73	31.0-36.5	20.15-23.73
	N	108+110N	54.0+55N	35.1+35.75N
MULTIPLY	2	4683-4843	2341.5-2421.5	1521.98-1573.98
DIVIDE	2,4	3645-8456	1822.5-4228	1184.63-2748.20
SQUARE ROOT	2	4336-4746	2168-2373	1409.20-1542.45
MOVE	N	91+49N	45.5+24.5N	29.58+15.93N
EXIT	-	60	30	19.5

```
                        ;                                                       ;
                        ;     DOUBLE PRECISION MINIMUM-MAXIMUM VALUE DETERMINATION      ;
                        ;                                                       ;
                        ;  <HL>   =  ADDRESS OF LEAST SIGNIFICANT BYTE OF LOWER BOUND  ;
                        ;  <HL+2> =  ADDRESS OF LEAST SIGNIFICANT BYTE OF UPPER BOUND  ;
                        ;  (BC)   =  NEW DATA                                         ;
                        ;  ZERO   =  1  INDICATES DATA WITHIN BOUNDS                  ;
                        ;  CY     =  1  INDICATES ND > UB                             ;
                        ;  CY     =  0  INDICATES ND < LB                             ;
                        ;   29 BYTES ROM + 57 BYTES FROM SIMPLE MATH PACKAGE, 95-264  ;
                        ;             MACHINE STATES AND 8 BYTES STACK USEAGE.      ;
                        ;                                                       ;
000537                  DMNMAX:
000537                          CALL    DCMPM           ; COMPARE ND TO LB
000537  315   033   001
000542                          RZ                      ; RETURN IF ND=LB
000542  310
000543                          JC      CKUB            ; CHECK FOR ND>UB
000543  332   153   001
000546                          MOV     M,C             ; ND<LB --- ESTABLISH NEW LB
000546  161
000547                          INX     H               ; STORE CURRENT ND IN LOCATION OF LB
000547  043
000550                          MOV     M,B
000550  160
000551                          DCX     H               ; RESTORE HL
000551  053
000552                          RET                     ; RETURN
000552  311
000553                  CKUB:   PUSH    H
000553  345
000554                          INX     H               ; POINT HL TO LSB OF UB BY INCREMENTING TWICE
000554  043
000555                          INX     H
000555  043
000556                          CALL    DCMPM           ; COMPARE ND TO UB
000556  315   033   001
000561                          JNC     DMNX            ; JUMP IF ND<=UB
000561  322   171   001
000564                          MOV     M,C             ; ELSE STORE NEW UB
000564  161
000565                          INX     H               ;
000565  043
000566                          MOV     M,B
000566  160
000567                          POP     H               ; RESTORE H
000567  341
000570                          RET                     ; RETURN
000570  311
000571                  DMNX:   XRA     A               ;
000571  257
000572                          POP     H               ;
000572  341
000573                          RET
000573  311
```

Program 4.4.2.2(1) Double Precision Minimum-Maximum Value Determination.

```
;                          MULTI-PRECISION                              ;
;       MINIMUM-MAXIMUM VALUE DETERMINATION SUBROUTINE                  ;
;                                                                       ;
; THIS SUBROUTINE COMPARES THE VALUE OF A NEW DATA POINT (ND)           ;
;   TO UPPER (UB) AND LOWER (LB) BOUNDS.  IF ND<LB, THE VAL-            ;
;    UE OF ND BECOMES THE NEW LOWER BOUND.  IF ND>LB, THE               ;
;     VALUE OF ND BECOMES THE NEW UPPER BOUND.     A FLAG               ;
;      IS SET TO INDICATE THE RESULT OF THE COMPARISONS                 ;
;       AND THE EXECUTION TIME OF THE SUBROUTINE.  ALL                  ;
;        NUMBERS ARE ASSUMED TO BE IN UNSIGNED BINARY                   ;
;         FORMAT.                                                       ;
;                                                                       ;
;         (A) = # BYTES / NUMBER.  SPECIFY TO GUARANTEE NO              ;
;               OVERFLOW ON ADDITION.                                   ;
;       (BC) = ADDRESS OF NEW DATA POINT'S LEAST SIG BYTE                ;
;       (HL) = ADDRESS OF UPPER BOUND LEAST SIGNIFICANT BYTE.           ;
;              THE LEAST SIG BYTE OF THE LOWER BOUND MUST BE            ;
;              STORED IMMEDIATELY ABOVE THE MOST SIGNIFICANT            ;
;              BYTE OF THE UPPER BOUND.                                 ;
;                                                                       ;
; ALL DATA WORDS IN MEMORY ARE ASSUMED TO BE STORED IN ASCEND-          ;
; MEMORY LOCATIONS BEGINNING AT THE ADDRESSES SPECIFIED ABOVE.          ;
;                                                                       ;
; THIS SUBROUTINE SAVES AND LATER RETURNS THE MACHINE STATUS.           ;
; TMCODE IS USED AS A FLAG BYTE.  ON RETURN, TMCODE=0 IF THE            ;
; NEW DATA IS BETWEEN THE BOUNDS.                                       ;
;                                                                       ;
;       TMCODE = 1 IF NEW DATA IS GREATER THAN THE UPPER BOUND          ;
;       TMCODE = 2 IF NEW DATA IS LESS THAN THE LOWER BOUND             ;
;                                                                       ;
; THIS SUBROUTINE DOES NOT ALTER THE VALUE OR LOCATION OF THE           ;
; NEW DATA POINT.       EXECUTION TIME IS A FUNCTION OF TMCODE          ;
; AND THE NUMBER OF BYTES PER NUMBER AS DEFINED BELOW:                  ;
;                                                                       ;
;       IF TMCODE = 0  THEN T = 269*A+414                               ;
;       IF TMCODE = 1  THEN T = 159*A+330                               ;
;       IF TMCODE = 2  THEN T = 269*A+510                               ;
;                      WHERE T = EXECUTION TIME IN STATES               ;

017776                  TMCODE = 17776          ; TIME CODE STORAGE BYTE LOCATION

000000                  MINMAX:
000000                          PUSH    PSW             ; SAVE CPU STATUS
000000  365
000001                          PUSH    B
000001  305
000002                          PUSH    D
000002  325
000003                          PUSH    H
000003  345
```

Program 4.4.2.2(2) Multi-Precision Minimum-Maximum Value Determination.

```
000004                                  MOV     E,A             ; SAVE NUMBER OF BYTES PER NUMBER IN E
000004  137
000005                                  CALL    MCMP            ; COMPARE NEW DATA (ND) POINT TO UPPER BOUND (UB)
000005  315     000     002
000010                                  DCR     D               ; DECREMENT D TWICE TO SET CONDITIONS FOR FLAG REGISTER D
000010  025
000011                                  DCR     D
000011  025
000012                                  JZ      OR              ; JUMP IF ND>UB
000012  312     037     000
000015                                  MVI     D,0             ; 0 ---> D
000015  026     000
000017                                  DAD     D               ; OTHERWISE (DE) + (HL) ---> (HL), NOW <HL> = LB
000017  031
000020                                  CALL    MCMP            ; COMPARE ND TO LOWER BOUND (LB)
000020  315     000     002
000023                                  DCR     D               ; DECREMENT TO TEST FLAG REGISTER D
000023  025
000024                                  JZ      UR              ; JUMP IF ND<LB, OTHERWISE ND IS WITHIN BOUNDS
000024  312     044     000
000027                                  MVI     A,0             ; 0 ---> A
000027  076     000
000031                                  STA     TMCODE          ; ZERO THE EXECUTION TIME REGISTER
000031  062     376     037
000034                                  JMP     EXIT            ; AND EXIT ROUTINE
000034  303     000     004
000037                          OR:
000037                                  MVI     A,1             ; 1 ---> A SINCE ND>UB
000037  076     001
000041                                  JMP     UR+2            ; CONTINUE AT UR+2
000041  303     046     000
000044                          UR:
000044                                  MVI     A,2             ; 2 ---> A SINCE ND<LB
000044  076     002
000046                                  STA     TMCODE          ; STORE A AS TIME CODE INDICATING EXECUTION
000046  062     376     037
                                                                ; TIME AND RESULT
000051                                  MOV     A,E             ; #BYTES/NUMBER ---> A
000051  173
000052                                  CALL    MOVE            ; MOVE ND TO LOCATION RESERVED FOR UB
000052  315     000     006
000055                                  JMP     EXIT            ; GO TO EXIT ROUTINE
000055  303     000     004
```

Program 4.4.2.2(2) (continued)

```
;                                                                       ;
;    DOUBLE PRECISION ZERO ORDER FLOATING APERTURE PREDICTOR             ;
;                                                                       ;
;  (BC) = NEW DATA POINT  [16 BITS]                                     ;
;  (HL) = ADDRESS OF LEAST SIGNIFICANT BYTE OF LOWER BOUND              ;
;         THE LEAST SIGNIFICANT BYTE OF THE UPPER BOUND SHOULD          ;
;         BE STORED AT THE MEMORY LOCATION ADDRESSED BY (HL)+2          ;
;  (DE) = ADDRESS OF TOLERANCE LEAST SIGNIFICANT BYTE                   ;
;                                                                       ;
;  ON RETURN:                                                           ;
;                Z   = 1 IF NEW DATA IS WITHIN TOLERANCE                ;
;                CY  = 0 IF NEW DATA IS LESS THAN LOWER BOUND            ;
;                CY  = 1 IF NEW DATA IS GREATER THAN UPPER BOUND         ;
;                FLAGS AND ACCUMULATOR ARE DESTROYED                    ;
;                                                                       ;
; 56 BYTES ROM + 34 BYTES FROM MATH SUBROUTINES, 6 BYTES STACK          ;
;                USEAGE AND 78-545 MACHINE STATES.                      ;
;                                                                       ;
000447                          DZOFAB:
000447                                  CALL    DCMPM           ; COMPARE ND TO LB
000447  315   033   001
000452                                  RZ                      ; ND=LB, DATA WITHIN TOLERANCE
000452  310
000453                                  JC      CKOR            ; JUMP IF ND>LB
000453  332   062   001
000456                                  PUSH    PSW             ; ND<LB, FLAG FOR TRANSMISSION, SAVE STATUS
000456  365
000457                                  JMP     NREF            ; SET UP NEW REFERENCE POINT AS ND
000457  303   075   001
000462                          CKOR:   INX     H               ; POINT (HL) TO LSB OF UB BY INCREMENTING TWICE
000462  043
000463                                  INX     H               ;
000463  043
000464                                  CALL    DCMPM           ; COMPARE ND TO UB
000464  315   033   001
000467                                  DCX     H               ; RESTORE (HL)
000467  053
000470                                  DCX     H               ;
000470  053
000471                                  JNC     SZRET           ; JUMP IF ND<= UB, DATA WITHIN TOLERANCE
000471  322   135   001
000474                                  PUSH    PSW             ; ND>UB, FLAG FOR TRANSMISSION SAVING STATUS
000474  365
000475                          NREF:   XCHG                    ; ESTABLISH ND AS NEW REFERENCE POINT
000475  353
000476                                  CALL    DSUBM           ; SUBTRACT TOLEREANCE FROM ND AND STORE AS NE LB
000476  315   063   000
000501                                  JNC     ENUB            ; JMP IF NO UNDERFLOW OCCURRED
000501  322   110   001
000504                                  XRA     A               ; ELSE CLEAR ACC
000504  257
000505                                  CALL    LMEM            ; UNDERFLOW --- SO SET LB=0
000505  315   130   001
```

Program 4.4.2.3(1) Double Precision Zero-Order Floating Aperture Predictor Subroutine.

```
000510                          ENUB:   INX     D               ; POINT DE TO LS BYTE OF UB BY INCREMENTING TWICE
000510  023
000511                                  INX     D               ;
000511  023
000512                                  CALL    DADDM           ; ADD TOLERANCE TO ND AND STORE AS NEW UB
000512  315     050     000
000515                                  JNC     EXZF            ; JUMP IF NO OVERFLOW OCCURRED
000515  322     125     001
000520                                  MVI     A,377           ; SET ACC TO ALL ONES
000520  076     377
000522                                  CALL    LMEM            ; OVERFLOW --- SO SET UB TO 65535
000522  315     130     001
000525                          EXZF:   POP     PSW             ; GET FLAGS BACK TO INDICATE RESULT OF ZOFAP
000525  361
000526                                  XCHG                    ; DE <---> HL
000526  353
000527                                  RET                     ; RETURN
000527  311
000530                          LMEM:   STAX    D               ; STORE CONSTANT IN ACC IN MEMORY
000530  022
000531                                  INX     D               ; INCREMENT POINTER
000531  023
000532                                  STAX    D               ; STORE SAME CONSTANT IN ADJACENT MEMORY LOCATION
000532  022
000533                                  DCX     D               ; RESTORE DE BY DECREMENTING
000533  033
000534                                  RET                     ; RETURN
000534  311
000535                          SZRET:  XRA     A               ; SET Z=1
000535  257
000536                                  RET                     ; RETURN
000536  311
```

Program 4.4.2.3(1) (continued)

based on the prediction that later data samples will be within a specified tolerance of the first sample. If this prediction is true, the data sample is considered redundant and should not be transmitted. The first sample which is not within the specified tolerance is flagged for transmission and becomes the new reference point for subsequent predictions. The result of the comparison is indicated by the states of the zero and carry flags after the program has been executed. The zero flag must be tested first to avoid incorrect results. A flow chart for the zero-order floating aperture predictor subroutine is illustrated in Figure 4.4.2.3(1). This program is 56 bytes long and uses three subroutines from the binary math package requiring an additional 34 bytes. Six bytes of RAM are required for stack operations. At present, 100-piece CMOS ROM prices, the incremental component cost of providing the zero-order floating aperture predictor subroutine and the required math package subroutines as part of a standard PDCP ROM, is approximately $7.21.* As shown in Section 4.4.3.2, the longest data formatting and transmitter control subroutine (GOES format) is 236 bytes long. Therefore, the double precision zero-order floating-aperture predictor algorithm and a transmitter subroutine would occupy only 32 percent of the capacity of an 8K bit ROM. The remaining 698 bytes of ROM should be sufficient to contain an executive program and several sensor data input and control subroutines. For example, the frequency sensor input subroutine [Program 4.4.1.1(1)] which is capable of serving eight different sensors requires only 36 additional bytes of ROM.

Program 4.4.2.3(2) is a multiprecision implementation of the zero-order floating-aperture predictor algorithm. A flow chart for this program is illustrated in Figure 4.4.2.3(2). Program execution times and

---

*For example, RCA's CDP1831CD 512 x 8 ROM with a 400 nsec access time and 0.5 mW typical quiescent power dissipation costs $41.00 each for the first 100 devices, including the mask charge. Therefore,

$$\Delta \text{ Cost} = \frac{41.00}{512 \text{ bytes}} \times 90 \text{ bytes} = \$7.21.$$

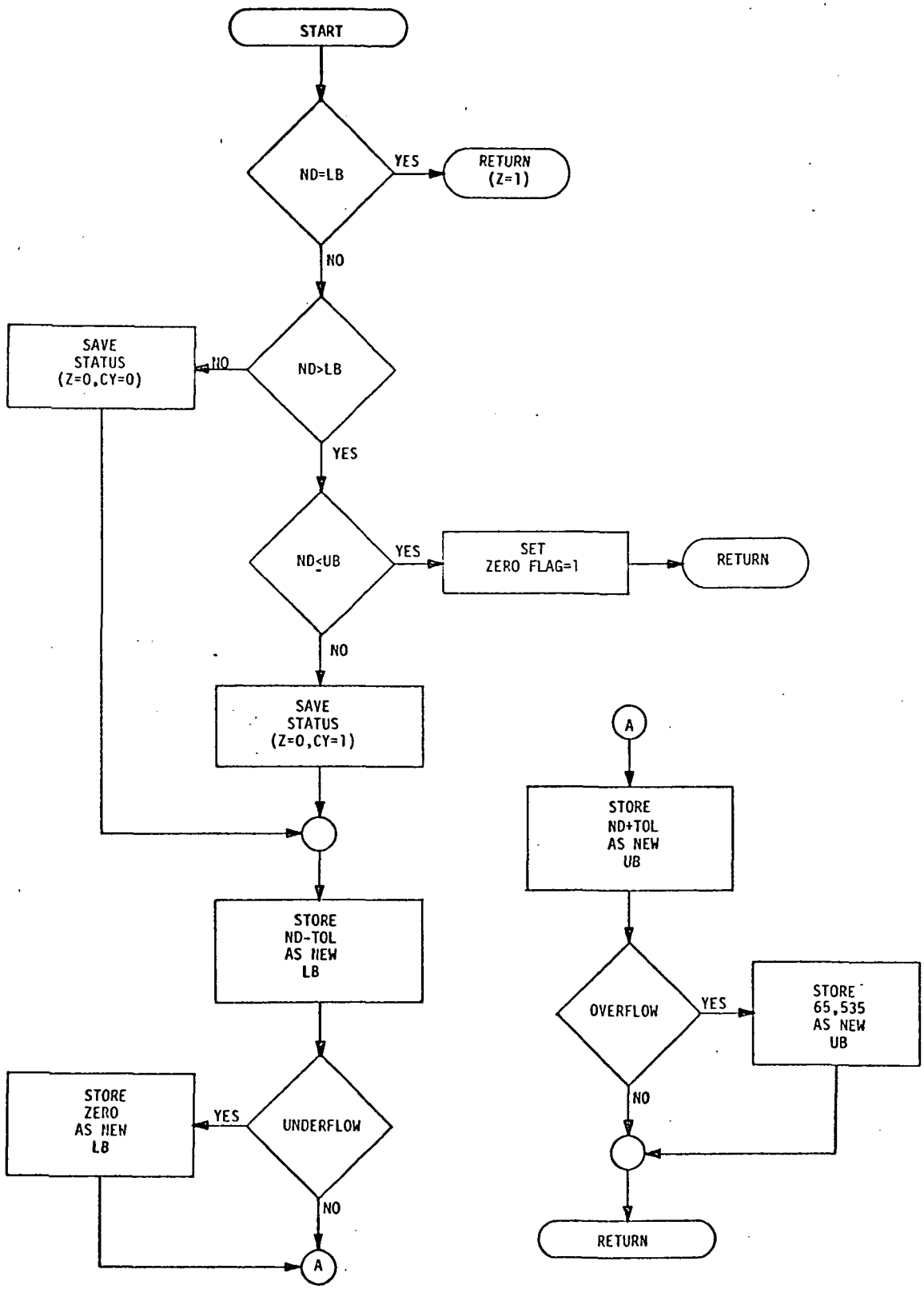


Figure 4.4.2.3(1) Flow Chart for the Double Precision Zero-Order Floating-Aperture Predictor Subroutine.

```
;       ZERO-ORDER FLOATING-APERTURE PREDICTOR SUBROUTINE
;
; THIS SUBROUTINE IMPLEMENTS THE ZERO-ORDER FLOATING-APERTURE
;   PREDICTOR ALGORITHM USING AN 8080A MICROPROCESSOR.  THIS
;   IS AN INFORMATION PRESERVING POLYNOMIAL DATA COMPRESSION
;     TECHNIQUE BASED ON THE PREDICTION THAT LATER SAMPLES
;       WILL BE WITHIN A SPECIFIED TOLERANCE OF THE FIRST
;         SAMPLE.  IF THIS PREDICTION IS TRUE, THE DATA
;           SAMPLE IS CONSIDERED REDUNDANT AND IS NOT
;            TRANSMITTED.  THE FIRST SAMPLE WHICH IS
;             NOT WITHIN THE SPECIFIED TOLERANCE IS
;              TRANSMITTED AND BECOMES THE NEW REF-
;               ERENCE POINT FOR SUBSEQUENT PREDIC-
;                TIONS.  ALL NUMBERS ARE ASSUMED TO
;                 BE IN N-BYTE, UNSIGNED BINARY
;                          FORMAT:
;
;       (A) = # BYTES / NUMBER.  SPECIFY TO GUARANTEE NO
;             OVERFLOW ON ADDITION.
;      (BC) = ADDRESS OF NEW DATA POINT'S LEAST SIG BYTE
;      (DE) = ADDRESS OF TOLERANCE LEAST SIGNIFICANT BYTE
;      (HL) = ADDRESS OF UPPER BOUND LEAST SIGNIFICANT BYTE.
;             THE LEAST SIG BYTE OF THE LOWER BOUND MUST BE
;             STORED IMMEDIATELY ABOVE THE MOST SIGNIFICANT
;             BYTE OF THE UPPER BOUND.
;
; ALL DATA WORDS IN MEMORY ARE ASSUMED TO BE STORED IN ASCEND-
; MEMORY LOCATIONS BEGINNING AT THE ADDRESSES SPECIFIED ABOVE.
;
; THIS SUBROUTINE SAVES AND LATER RETURNS THE MACHINE STATUS.
; TMCODE IS USED AS A FLAG BYTE.  ON RETURN, TMCODE=0 IF THE
; NEW DATA POINT WAS REDUNDANT.  TMCODE>0 IF THE NEW DATA POINT
;  IS NOT REDUNDANT AND THEREFORE HAS BEEN ESTABLISHED AS THE
;    NEW REFERENCE POINT.  SPECIFICALLY:
;
;       TMCODE = 1 IF NEW DATA IS GREATER THAN THE UPPER BOUND
;                  OF THE REFERENCE PLUS TOLERANCE
;       TMCODE = 2 IF NEW DATA IS LESS THAN THE LOWER BOUND
;                    OF THE REFERENCE MINUS TOLERANCE
;
; THIS SUBROUTINE DOES NOT ALTER THE VALUE OR LOCATION OF THE
; NEW DATA POINT.     EXECUTION TIME IS A FUNCTION OF TMCODE
; AND THE NUMBER OF BYTES PER NUMBER AS DEFINED BELOW:
;
;       IF TMCODE = 0  THEN T = 220*A+414
;       IF TMCODE = 1  THEN T = 320*A+771
;       IF TMCODE = 2  THEN T = 430*A+901
;                      WHERE T = EXECUTION TIME IN STATES
;                            A = NUMBER OF BYTES PER NUMBER
;
```


Program 4.4.2.3(2) Zero-Order Floating-Aperture Predictor.

```
                                        ORG     11000   ; START PROGRAM AT 4.5 K POINT
017777                                  TMCODE= 17777   ; TIME CODE STORAGE BYTE LOCATION


011000                          ZOFAB:
011000                                  PUSH    PSW     ; SAVE CPU STATUS
011000  365
011001                                  PUSH    B
011001  305
011002                                  PUSH    D
011002  325
011003                                  PUSH    H
011003  345
011004                                  MOV     E,A     ; SAVE NUMBER OF BYTES PER NUMBER IN E
011004  137
011005                                  CALL    MCMP    ; COMPARE NEW DATA (ND) POINT TO UPPER BOUND (UB)
011005  315     042     020
011010                                  DCR     D       ; DECREMENT D TWICE TO SET CONDITIONS FOR FLAG REGISTER D
011010  025
011011                                  DCR     D
011011  025
011012                                  JZ      OR      ; JUMP IF ND>UB
011012  312     037     022
011015                                  MVI     D,0     ; 0 ----> D
011015  026     000
011017                                  DAD     D       ; OTHERWISE (DE) + (HL) ----> (HL), NOW <HL> = LB
011017  031
011020                                  CALL    MCMP    ; COMPARE ND TO LOWER BOUND (LB)
011020  315     042     020
011023                                  DCR     D       ; DECREMENT TO TEST FLAG REGISTER D
011023  025
011024                                  JZ      UR      ; JUMP IF ND<LB, OTHERWISE ND IS WITHIN TOLERANCE
011024  312     044     022
011027                                  MVI     A,0     ; 0 ----> A
011027  076     000
011031                                  STA     TMCODE  ; ZERO THE EXECUTION TIME REGISTER
011031  062     377     037
011034                                  JMP     EXIT    ; AND EXIT ROUTINE
011034  303     147     020
011037                          OR:
011037                                  MVI     A,1     ; 1 ----> A SINCE ND>UB
011037  076     001
011041                                  JMP     UR+2    ; CONTINUE AT UR+2
011041  303     046     022
```


Program 4.4.2.3(2) (continued)

```
011044                          UR:.
011044                                  MVI     A,2     ; 2 ---> A SINCE ND<LB
011044  076     002
011046                                  STA TMCODE.     ; STORE A AS TIME CODE INDICATING EXECUTION
011046  062     377     037
                                        ; TIME AND RESULT
011051                                  MOV     A,E     ; #BYTES/NUMBER ---> A
011051  173
011052                                  POP     H       ; ADDRESS OF UB ---> (HL)
011052  341
011053                                  PUSH    H       ; SAVE ADDRESS ON STACK
011053  345
011054                                  CALL    MOVE    ; MOVE ND TO LOCATION RESERVED FOR UB
011054  315     130     020
011057                                  MVI     D,0     ; 0 ---> D
011057  026     000
011061                                  DAD     D       ; (DE) + (HL) ---> (HL) SO <HL> = LB
011061  031
011062                                  CALL     MOVE   ; MOVE ND TO LOCATION RESERVED FOR LB
011062  315     130     020
011065                                  MOV     B,H     ; TRANSFER THE UPPER AND LOWER BYTES OF LB
011065  104
011066                                  MOV     C,L     ; ADDRESS TO (BC)
011066  115
011067                                  INX     SP      ; INCREMENT STACK POINTER TWICE TO POINT TO
011067  063
011070                                  INX     SP      ; LOCATION CONTAINING THE ADDRESS OF TOL
011070  063
011071                                  POP     H       ; GET TOL ADDRESS TO HL
011071  341
011072                                  PUSH    H       ; SAVE ADDRESS OF TOL ON STACK
011072  345
011073                                  DCX     SP      ; DECREMENT THE STACK POINTER TWICE TO
011073  073
011074                                  DCX     SP      ; RETURN TO ORIGINAL ADDRESS
011074  073
011075                                  CALL    MSUB    ; SUBTRACT TOL FROM ND AND STORE AS NEW LB
011075  315     021     020
011100                          EUB:    POP     B       ; ADDRESS OF LB ---> BC
011100  301
011101                                  PUSH    B       ; SAVE LB ON STACK
011101  305
011102                                  CALL     MADD   ; ADD TOL TO ND AND STORE AS THE NEW UB
011102  315     000     020
011105                                  JMP     EXIT    ; RESTORE MACHINE STATUS AND RETURN TO
011105  303     147     020
                                        ; EXECUTIVE PROGRAM.


011110                                  END
```


Program 4.4.2.3(2) (continued)

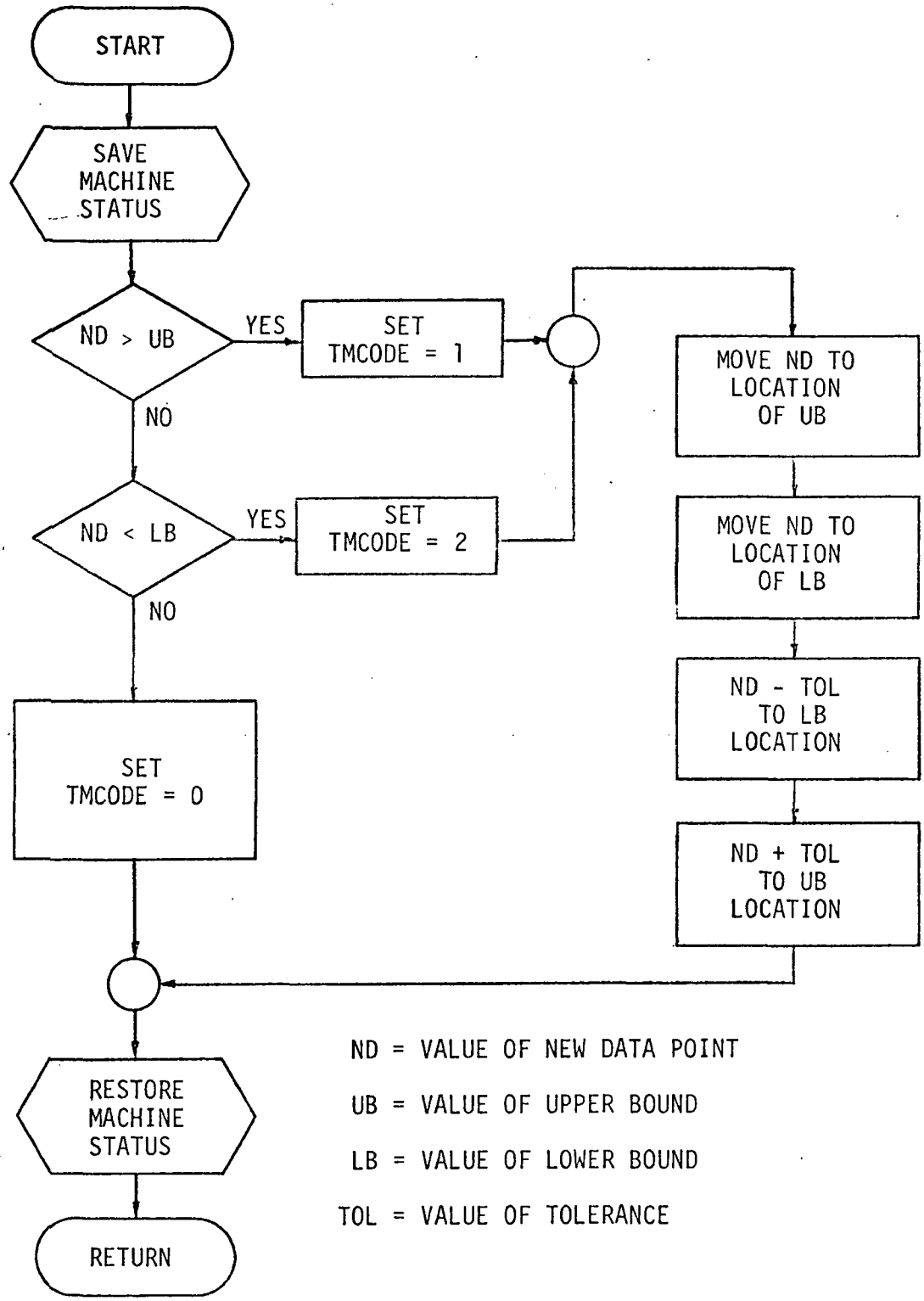


Figure 4.4.2.3(2) Flow Chart for the Multiprecision Zero-Order Floating Aperture Predictor Subroutine.

the results of the multiprecision zero-order floating-aperture predictor subroutine are indicated by the value returned in the memory location called TMCODE. This program is 71 bytes long and uses four subroutines from the multiprecision binary math package for a minimum storage requirement of 178 bytes. Twenty bytes of RAM are required for stack operations.

4.4.2.4 Data Average - Data averaging is one useful form of data preprocessing which can easily be performed by a PDCP. Normally two distinct subroutines will be used to obtain the average value of a set of N data points. A data accumulation subroutine such as Program 4.4.2.4(1) will be called each time a data sample is obtained. When the average value of the data is desired, a final data averaging subroutine will be called. As illustrated by the flow chart in Figure 4.4.2.4(1), the data accumulation subroutine will simply increment the sample count (N) and add the new sample to the previous sum of samples. Program 4.4.2.4(1) can process up to 65,535 samples of 16-bit sensor data while using only 18 bytes of ROM and 10 bytes of stack.

Program 4.4.2.4(2) is an example of a data averaging subroutine. A flow chart for this program is shown in Figure 4.4.2.4(2). The data averaging subroutine calls the short multiprecision divide routine to divide the accumulated sum of samples by the number of samples, N. The resultant average is returned in register pair DE. Only 24 bytes of ROM and 18 bytes of stack are required for this subroutine. Therefore, data averaging can be accomplished by the PDCP at a cost of only 42 bytes of ROM and 18 bytes of stack in addition to the general purpose binary math subroutines.

4.4.2.5 Mean, Variance, and Standard Deviation - Much more information can be obtained from the mean or average of a number of data samples if the variance and standard deviation are also known. Programs 4.4.2.5(1) and 4.4.2.5(2) can be used in conjunction with the data accumulation subroutine [Program 4.4.2.4(1)] to calculate this information aboard the PDCP. Program 4.4.2.5(1) is called each time a data sample is obtained. This program prepares the data for the mean, variance, and standard deviation subroutine by calculating the number of samples, the

```
                        ;                                                       ;
                        ;                DATA ACCUMULATION SUBROUTINE           ;
                        ;                                                       ;
                        ;       DATA = (BC) IS ADDED TO PREVIOUSLY STORED DATA  ;
                        ;      THE STORAGE BLOCK IS SIX BYTES BEGINNING AT (HL) ;
                        ;    THE FIRST TWO BYTES, M<HL> AND M<HL+1> ARE THE NUMBER ;
                        ;    OF DATA BYTES ADDED IN.  THE FOUR BYTES FROM (HL+2) TO ;
                        ;    (HL+5) CONTAIN THE RUNNING SUM OF THE DATA ACCUMULATED. ;
                        ;     LEAST SIGNIFICANT BYTES OCCUPY THE SMALLER ADDRESSES ;
                        ;                ALL CPU STATUS IS PRESERVED.           ;
                        ;                                                       ;
                        ;  18 BYTES ROM + 20 BYTES FROM SMADD ROUTINE, 10 BYTES STACK ;
                        ;                USEAGE AND 245 MACHINE STATES.         ;
                        ;                                                       ;
000574                  DACC:
000574                          PUSH    PSW             ; SAVE CPU STATUS
000574  365
000575                          PUSH    H               ;
000575  345
000576                          MOV     A,M             ; # BYTES PREVIOUSLY STORED ----> A
000576  176
000577                          ADI     1               ; INCREMENT A
000577  306     001
000601                          MOV     M,A             ; SAVE A IN MEMORY
000601  167
000602                          INX     H               ; INCREMENT MEMORY POINTER
000602  043
000603                          MOV     A,M             ; GET MS BYTE OF DATA BYTES STORED
000603  176
000604                          ACI     0               ; INCREMENT IF OVERFLOW FROM LS BYTE
000604  316     000
000606                          MOV     M,A             ; RESTORE MS BYTE TO MEMORY
000606  167
000607                          INX     H               ; POINT HL TO DATA SUM BLOCK
000607  043
000610                          CALL    SMADD           ; ADD IN NEW DATA
000610  315     000     000
000613                          POP     H               ; RESTORE CPU STATUS
000613  341
000614                          POP     PSW             ;
000614  361
000615                          RET                     ; RETURN
000615  311
```

Program 4.4.2.4(1) Data Accumulation Subroutine.

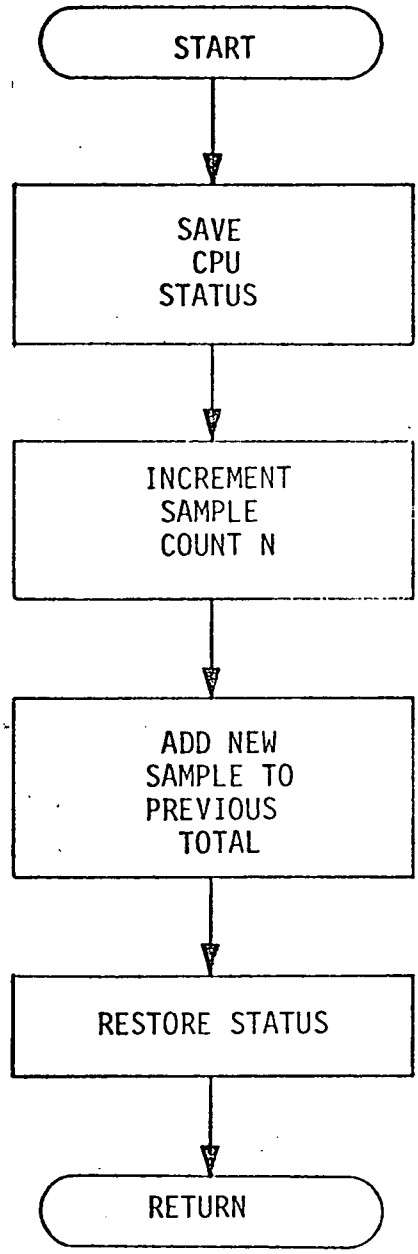


Figure 4.4.2.4(1) Flow Chart for the Data Accumulation Subroutine.

```
                        ;                    DATA AVERAGING SUBROUTINE                    ;
                        ;                                                                 ;
                        ;          (M<HL>) = NUMBER OF DATA POINT ACCUMULATED             ;
                        ;                (M<HL+2>) = ACCUMULATED SUM                      ;
                        ;             OF UP TO 32 BIT RESULT BEFORE OVERFLOW              ;
                        ;                                                                 ;
                        ;  AVERAGE OF ACCUMULATED DATA IS RETURNED IN DE (ROUNDED OFF)    ;
                        ;                                                                 ;
                        ; 24 BYTES ROM + 71 BYTES FROM DFDIV ROUTINE, 10 BYTES OF STACK  ;
                        ;                  USEAGE AND 3727-8403 MACHINE STATES.           ;
                        ;                                                                 ;
000616                  AVG:    PUSH    PSW             ; SAVE MACHINE STATUS
000616  365
000617                          PUSH    B               ;
000617  305
000620                          PUSH    H               ;
000620  345
000621                          MOV     C,M             ; #BYTES TO BE AVERAGED--->BC
000621  116
000622                          INX     H               ;
000622  043
000623                          MOV     B,M             ;
000623  106
000624                          INX     H               ;
000624  043
000625                          MOV     E,M             ; ACCUMULATED SUM OF
000625  136
000626                          INX     H               ; DATA ----> HLDE
000626  043
000627                          MOV     D,M             ;
000627  126
000630                          INX     H               ;
000630  043
000631                          PUSH    D               ;
000631  325
000632                          MOV     E,M             ;
000632  136
000633                          INX     H               ;
000633  043
000634                          MOV     D,M             ;
000634  126
000635                          XCHG                    ;
000635  353
000636                          POP     D               ;
000636  321
000637                          CALL    DFDIV           ; DIVIDE ACCUMULATED SUM BY # SAMPLES
000637  315     207     000
000642                  EXD:    POP     H               ; RESTORE MACHINE STATUS
000642  341
000643                          POP     B               ;
000643  301
000644                          POP     PSW             ;
000644  361
000645                          RET                     ; RETURN
000645  311
```

Program 4.4.2.4(2) Data Averaging Subroutine.

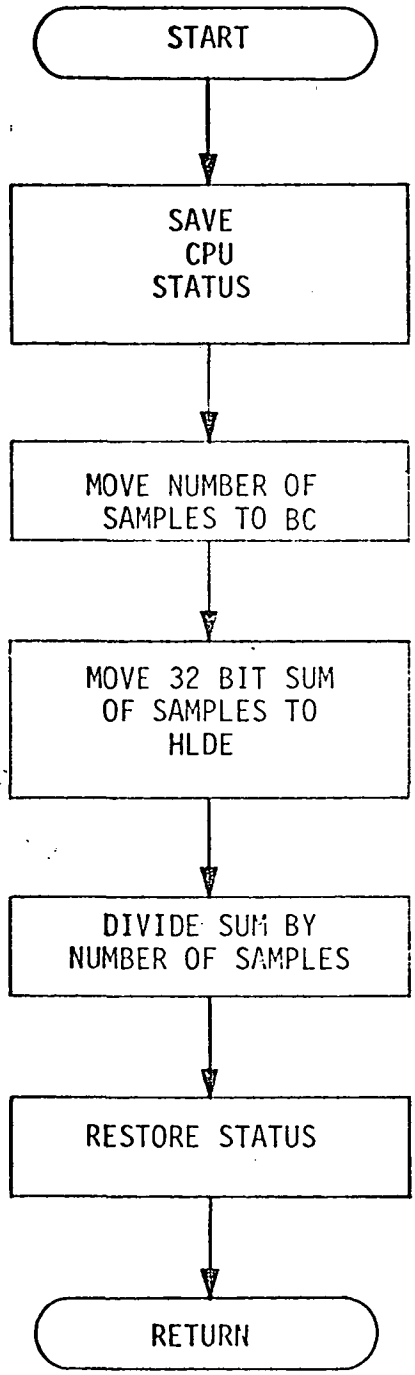


Figure 4.4.2.4(2) Flow Chart for the Data Averaging Subroutine.

```
                        ;  DATA PREPARATION FOR MEAN, VARIANCE AND STANDARD DEVIATION   ;
                        ;                                                               ;
                        ;  (BC)       =    DATA  (2 BYTES)                            ;
                        ;  (M<HL>)    =    N  (2 BYTES)                               ;
                        ;  (M<HL+2>)  =    SUM OF SAMPLES  (4 BYTES)                  ;
                        ;  (M<HL+6>)  =    SCRATCH  (4 BYTES)                         ;
                        ;  (M<HL+10>) =    SUM OF SQUARED SAMPLES                     ;
                        ; 39 BYTES ROM ; 122 BYTES FROM MATH PACKAGE, 5301-5541 STATES ;
                        ;                    AND 20 BYTES STACK USEAGE.                 ;
                        ;                                                               ;
000646                  DPMSD:
000646                          PUSH    PSW             ; SAVE STATUS
000646  365
000647                          PUSH    B
000647  305
000650                          PUSH    D
000650  325
000651                          PUSH    H
000651  345
000652                          CALL    DACC            ; INCREMENT SAMPLE COUNT AND ADD SAMPLE TO PREVIOUS SAMPL
000652  315     174     001
000655                          LXI     D,6             ; 6--->DE
000655  021     006     000
000660                          DAD     D               ; ADD (DE) TO HL TO POINT HL AT SCRATCH AREA
000660  031
000661                          MOV     E,C             ; DUPLICATE SAMPLE IN DE
000661  131
000662                          MOV     D,B             ;
000662  120
000663                          CALL    DPMULT          ; MULTIPLY (DE)*(BC) TO SQUARE SAMPLE
000663  315     076     000
000666                          MOV     C,M             ; PLACE LS TWO BYTES OF SQUARE IN BC
000666  116
000667                          INX     H               ;
000667  043
000670                          MOV     B,M             ;
000670  106
000671                          INX     H               ; PLACE TWO MS BYTES OF SQUARE IN DE
000671  043
000672                          MOV     E,M             ;
000672  136
000673                          INX     H               ;
000673  043
000674                          MOV     D,M             ;
000674  126
000675                          INX     H               ; POINT HL TO STORAGE AREA FOR SUM OF SQUARED SAMPLES
000675  043
```

Program 4.4.2.5(1) Data Preparation Subroutine for Mean, Variance, and Standard Deviation.

```
000676
000676  315     000     000             CALL    SMADD           ; ADD LS TWO BYTES OF SQUARED SAMPLE TO CURRENT SUM
000701
000701  043                             INX     H               ; POINT HL TO TWO MS BYTES OF CURRENT SUM OF SQUARES
000702
000702  043                             INX     H               ; BY INCREMENTING TWICE
000703
000703  113                             MOV     C,E             ; PLACE MS TWO BYTES OF CURRENT SQUARE IN BC
000704
000704  102                             MOV     B,D
000705
000705  135                             MOV     E,L             ; POINT DE TO TWO MS BYTES OF CURRENT SUM OF SQUARES
000706
000706  124                             MOV     D,H
000707
000707  315     050     000             CALL    DADDM           ; ADD TWO MS BYTES OF CURRENT SQUARE TO SUM OF SQUARES
000712
000712  303     104     002             JMP     EXIT            ; RESTORE STATUS AND RETURN TO CALLING POINT
```

Program 4.4.2.5(1) (continued)

```
;                                                                       ;
;       MEAN, VARIANCE, AND STANDARD DEVIATION SUBROUTINE               ;
;                                                                       ;
; THIS SUBROUTINE CALCULATES THE MEAN, VARIANCE AND STANDARD            ;
;   DEVIATION OF DATA WHICH HAS BEEN PREPROCESSED BY THE DATA           ;
; PREPARATION SUBROUTINE. THE PROGRAM EXPECTS THE FOLLOWING IN-         ;
; FORMATION IN MEMORY AS SPECIFIED BELOW:                               ;
;                                                                       ;
;  (M<HL>)   =    N                      (M<HL+6>)  = SCRATCH          ;
;  (M<HL+2>) = SUM OF SAMPLES      (M<HL+10>) = SUM OF SQUARES          ;
;                                                                       ;
; THE MEAN, NUMBER OF SAMPLES, VARIANCE, AND STANDARD DEVIATION         ;
; ARE RETURNED IN MEMORY AS SPECIFIED BELOW:                            ;
;                                                                       ;
;                        (M<HL-2>)  =  MEAN                             ;
;                        (M<HL>)    =  NUMBER OF SAMPLES                ;
;                        (M<HL+2>)  =  VARIANCE                         ;
;                        (M<HL+4>)  =  STANDARD DEVIATION               ;
;                                                                       ;
; 93 BYTES + 272 BYTES FROM ADVANCED MATH PACKAGE, 20930-05934          ;
;           MACHINE STATES AND 28 BYTES STACK USEAGE.                   ;
;                                                                       ;
000715                  MVSD:
000715                          PUSH    PSW             ; SAVE ALL STATUS
000715  365
000716                          PUSH    B
000716  305
000717                          PUSH    D
000717  325
000720                          PUSH    H
000720  345
000721                          CALL    AVG             ; AVERAGE OF SAMPLES WILL BE RETURNED IN DE
000721  315     216     001
000724                          DCX     H               ;
000724  053
000725                          MOV     M,D             ; STORE AVERAGE AT M<HL-2>
000725  162
000726                          DCX     H               ;
000726  053
000727                          MOV     M,E             ;
000727  163
000730                          INX     H               ; POINT HL TO SUM OF SAMPLES BY INCREMENTING 4 TIMES
000730  043
000731                          INX     H
000731  043
000732                          INX     H
000732  043
000733                          INX     H
000733  043
000734                          MOV     E,M             ; LS BYTE OF SUM OF SAMPLES TO E AND C
000734  136
000735                          MOV     C,M             ;
000735  116
000736                          INX     H               ;
000736  043
000737                          MOV     D,M             ; MS BYTE OF SUM OF SAMPLES TO D AND B
000737  126
```

Program 4.4.2.5(2) Mean, Variance, and Standard Deviation Subroutine.

```
000740                          MOV     D,M             ; POINT HL TO SCRATCH AREA BY INCREMENTING 3 TIMES
000740  106
000741                          INX     H               ;
000741  043
000742                          INX     H               ;
000742  043
000743                          INX     H               ;
000743  043
000744                          CALL    DPMULT          ; MULTIPLY DE*BC TO SQUARE SUM OF SAMPLES
000744  315     076     000
000747                          CALL    MVSD1           ;
000747  315     022     002
000752                          CALL    DPDIV           ; (HLDE)/(BC) ----> DE
000752  315     207     000
000755                          POP     H               ; POINT HL TO N
000755  341
000756                          PUSH    H               ; SAVE HL AGAIN
000756  345
000757                          LXI     B,10.           ; 10. ----> BC
000757  001     012     000
000762                          DAD     B               ; ADD 10. TO HL TO POINT TO SUM OF SQUARED SAMPLES
000762  011
000763                          MOV     B,D             ; QUOTIENT TO BC
000763  102
000764                          MOV     C,E             ;
000764  113
000765                          CALL    SMSUB           ; SUBTRACT (BC) FROM (M<HL>) TO FORM (N-1)*VARIANCE
000765  315     024     000
000770                          CALL    MVSD1           ; MOVE RESULT TO HLDE AND # OF SAMPLES TO BC
000770  315     022     002
000773                          DCX     B               ; N-1 ----> BC
000773  013
000774                          CALL    DPDIV           ; DIVIDE DIFFERENCE MY N-1 TO GET VARIANCE
000774  315     207     000
000777                          POP     H               ; GET ADDRESS OF N
000777  341
001000                          PUSH    H               ; SAVE HL AGAIN
001000  345
001001                          INX     H               ; INCREMENT HL TWICE
001001  043
001002                          INX     H               ;
001002  043
001003                          MOV     M,E             ; STORE VARIANCE ABOVE #SAMPLES IN MEMORY
001003  163
001004                          INX     H               ;
001004  043
001005                          MOV     M,D             ;
001005  162
001006                          XCHG                    ;
001006  353
001007                          CALL    SQRT            ; SQR'L OF VARIANCE TO ACCUMULATOR
001007  315     307     000
001012                          INX     D               ;
001012  023
001013                          STAX    D               ; STORE STANDARD DEVIATION IN MEMORY
001013  022
001014                          INX     D               ;
```

Program 4.4.2.5(2) (continued)

```
001014  023
001015                          XRA     A               ;
001015  257
001016                          STAX    D               ;
001016  022
001017                          JMP     EXIT            ; RESTORE STATUS AND RETURN TO CALLING PROGRAM
001017  303     104     002
001022                  MVSD1:                          ; SUBROUTINE TAKES 4 BYTES FROM MEMORY BEGINNING AT
                                                        ; <HL> AND PUTS THEM IN HLDE.  ALSO LOADS BC WITH
                                                        ; (M<LAST VALUE ON STACK>)
001022                          MOV     E,M             ; MOVE LS TWO BYTES OF RESULT TO DE
001022  136
001023                          INX     H               ;
001023  043
001024                          MOV     D,M             ;
001024  126
001025                          INX     H               ;
001025  043
001026                          PUSH    D               ; SAVE LS TWO BYTES
001026  325
001027                          MOV     E,M             ; MOVE MS TWO BYTES TO DE
001027  136
001030                          INX     H               ;
001030  043
001031                          MOV     D,M             ;
001031  126
001032                          INX     SP              ; POINT SP TO STACK AREA WHERE THE ADRESS
001032  063
001033                          INX     SP              ; OF THE NUMBER OF SAMPLES IS STORED BY INCREMENTING
001033  063
001034                          INX     SP              ; 4 TIMES
001034  063
001035                          INX     SP              ;
001035  063
001036                          POP     H               ; GET ADDRESS OF NUMBER OF SAMPLES
001036  341
001037                          PUSH    H               ; RESAVE IT
001037  345
001040                          DCX     SP              ; RESTORE SP BY DECREMENTING 4 TIMES
001040  073
001041                          DCX     SP              ;
001041  073
001042                          DCX     SP              ;
001042  073
001043                          DCX     SP              ;
001043  073
001044                          MOV     C,M             ; MOVE # SAMPLES TO BC
001044  116
001045                          INX     H               ;
001045  043
001046                          MOV     B,M             ;
001046  106
001047                          POP     H               ; PUT 2 MS BYTES OF RESULT INTO HL
001047  341
001050                          XCHG                    ;
001050  353
001051                          RET                     ; RETURN
```

Program 4.4.2.5(2) (continued)

sum of the samples, and the sum of the squared samples. A flow chart for Program 4.4.2.5(1) is provided in Figure 4.4.2.5(1).

Program 4.4.2.5(2) calculates the mean, variance, and standard deviation of the data using the formulas

$$\text{MEAN} = \frac{1}{n} \sum_{i=1}^{n} X_i \ ,$$

$$\text{Variance } (S^2) = \frac{1}{n-1} \left[\sum_{i=1}^{n} X_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} X_i \right)^2 \right], \text{ and}$$

$$\text{Standard Deviation}(S) = \sqrt{S^2} \ .$$

A PDCP with this capability would typically calculate and transmit the mean and variance of sensor data acquired between transmissions rather than transmitting a large number of data points. This will significantly reduce the load on the satellite data collection system by reducing bandwidth and data storage requirements. In addition, this data preprocessing operation provides the user with immediate knowledge of the average size of the sample values and the dispersion of the samples about the mean.

A flow chart for Program 4.4.2.5(2) is shown in Figure 4.4.2.5(2). The data accumulation; data preparation for mean, variance, and standard deviation; and mean, variance, and standard deviation subroutines require a total of 150 bytes of program storage and 28 bytes of stack.

4.4.3 Data Formatting and Transmitter Control

Data formatting and transmitter control are two of the four basic tasks performed by the hardwired control logic currently used in most DCP designs. Special emphasis has been placed on developing data formatting and transmitter control subroutines for each of the major data collection satellites because these are essential PDCP tasks which require

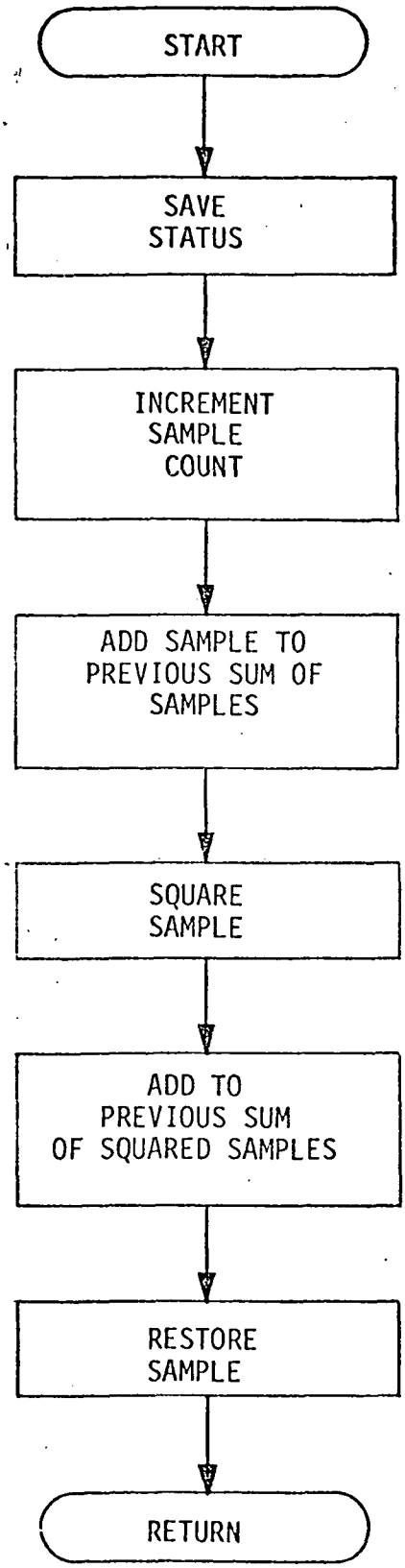


Figure 4.4.2.5(1) Flow Chart for Data Preparation for Mean, Variance, and Standard Deviation Subroutine.

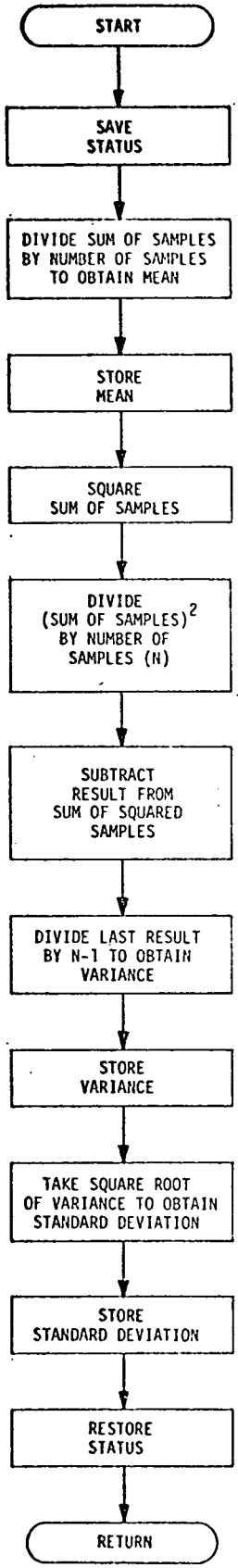


Figure 4.4.2.5(2) Flow Chart for Mean, Variance, and Standard Deviation Subroutine.

precise timing. Normally, both data formatting and transmitter control will be provided by a single subroutine in order to reduce the amount of RAM required for data storage. For example, in the GOES data transmission format an eight-bit data byte is encoded as two 11-bit ASCII words. Each ASCII word is then Manchester encoded before transmission, Thus, as many as 44 bites of RAM would be required to store only eight bits of data. However, if the microprocessor is sufficiently fast, the RAM allocation for data storage can be reduced 82 percent by performing both the ASCII and Manchester encoding in real time during transmission.

The transmitter control programs illustrated in this section assume transmitter modules which operate in basically the same manner as the Landsat-GOES convertible transmitter designed by Ball Brothers Corporation [6]. This transmitter module was chosen because control signal specifications were readily available. Note that the programs presented in this section can easily be modified to function with any reasonable transmitter design.

Ball Brothers' transmitter control signals INTEGRATE, TRANSMITTER ENABLE, and ±15V POWER ON are designated as INT, XMITE, and XPWR, respectively. To demonstrate the versatility of a PDCP, additional control signals which could select Landsat (LS), GOES (GO), TIROS-N (TN), or TWERLE (TW) transmitter modes are provided. Normally, a particular PDCP would be required to transmit data to only one type of satellite, and thus a simple modular transmitter designed specifically for that satellite could be used. In this case, control signals LS, GO, TN, and TW would not be required. However, if an electrically convertible transmitter is available and a PDCP is required to transmit data to more than one type of satellite, the LS, GO, TN, and TW signals would enable the correct transmitter mode.

The 8080A programs listed in this section assign transmitter control to output port XCNTR (Port 1). Biphase data to the transmitter appears on the least significant bit of output port XMIT (Port Ø, see Section 5.1.1.4). Bit assignments for the individual control signals on port XCNTR are illustrated in Figure 4.4.3(1). The function of each

OUTPUT PORT XCNTR

7	6	5	4	3	2	1	Ø	BIT NUMBER
TN	XPWR	XMITE	GO	LS	$\overline{\text{INT}}$	TW	(UNUSED)	ASSIGNMENT

Figure 4.4.3(1) Bit Assignments for Transmitter Control Port XCNTR.

control signal is specified in Table 4.4.3(1). As an example of microprocessor control of the transmitter, consider the 8080A assembly language instructions

```
MVI A,124

OUT XCNTR .
```

The first instruction moves the octal byte 124 into the accumulator, and the second instruction causes the contents of the accumulator to be latched into output port XCNTR. As a result, power is applied to the transmitter, GOES mode is selected, and the integrator is disabled as specified by the binary code ØlØlØlØØ.

Timing is an important consideration in transmitter subroutines. Correct reception of data at the satellite depends upon the self-clocking characteristic of the Manchester code. The transmission rate must be held constant during each transmission, and the Ø → 1 or 1 → Ø transitions representing data must occur precisely at the middle of each bit time. In addition, the transmission rate must be held within a reasonably tight tolerance over the operational life of the PDCP, which can be greater than one year. Because some variation in the frequency of the PDCP master clock must be expected, the transmitter subroutines should not be allowed to contribute to the total error budget. The entire error budget can then be applied to the system oscillator so that the cost of the clock will be minimized.

Since microprocessor controlled operations can only occur at discrete time intervals defined by the system clock, the clock frequency must be chosen to provide an integer number of states during each phase of a data bit. This technique has been employed in the design of the model PDCP (see Section 5.1.1.1). A low frequency clock was chosen so that the microprocessor could be interfaced with inexpensive, low speed memory without the need for a wait-cycle generator. Further, the specific 600-KHz CPU clock was chosen so that an integer number of states will occur within each phase of the Manchester encoded data for all typical

TABLE 4.4.3(1)

TRANSMITTER CONTROL SIGNALS

Signal	Bit Number	Function
(UNUSED)	Ø	UNUSED BIT provides don't care condition to minimize the number of instructions required to initialize the transmitter data port (Port XMIT)
TW	1	Selects TWERLE transmitter mode
$\overline{\text{INT}}$	2	$\overline{\text{INTEGRATE}}$ inhibits carrier modulation and thus control clear carrier transmission
LS	3	Selects Landsat transmitter mode
GO	4	Selects GOES transmitter mode
XMITE	5	Enables the transmitter RF signal
XPWR	6	Controls power supply to transmitter
TN	7	Selects TIROS-N transmitter mode

transmission rates. This restriction on the microprocessor clock does not significantly reduce the potential speed of the microprocessor. For example, only a 0.225 percent increase in the theoretical minimum cycle time of the higher speed 8080A-1 microprocessor is required to satisfy the restriction. Table 4.4.3(2) lists the transmission rates for each major data collection satellite system and the corresponding number of states within each phase of the Manchester-encoded data stream for an 8080A with a 600-KHz clock and for an 8080A-1 with a 3.07-MHz clock. Note that the microprocessor must output to port XMIT at a rate equivalent to twice the data rate.

TABLE 4.4.3(2)

TRANSMISSION RATES FOR THE LANDSAT, GOES, TWERLE, AND TIROS-N DATA COLLECTION SYSTEMS

System	Transmission Rate (Bits Per Second)	Number of States Between Phases of the Manchester-Encoded Data Stream	
		8080A µP 600-KHz Clock	8080A-1 µP 3070-KHz Clock
TWERLE	100	3000	15,350
GOES	100	3000	15,350
TIROS-N	400	750	7,675
Landsat	5000	60	307

4.4.3.1 TWERLE Format Transmitter Subroutine - Specifications for the TWERLE data transmission sequence are shown in Table 4.4.3.1(1). Program 4.4.3.1(1) is a listing of the TWERLE format transmitter routine written for the UT PDCP. This subroutine provides complete control of the transmitter and generates a Manchester-encoded data stream at a rate of precisely 100 bits per second. A simplified flow chart for Program 4.4.3.1(1) is illustrated in Figure 4.4.3.1(1), and a detailed flow chart for the data transmission block is presented in Figure 4.4.3.1(2).

TABLE 4.4.3.1(1)

TWERLE DATA TRANSMISSION SEQUENCE
SPECIFICATIONS

Transmission Interval:	1 second nominal
Transmission Rate:	100 bits per second
Coding:	Manchester encoded with a 0 → 1 transition representing a 1

Transmission Sequence:

1. Transmitter power-up followed by a 1 second warm-up delay
2. Clear carrier transmission for 0.32 to 0.36 seconds
3. Data transmission
   - a. Bit synchronization code (10101010) (8 bits)
   - b. Frame synchronization code (110101100000) (12 bits)
   - c. Address code (assigned to user) (10 bits)
   - d. Mode bits (2 MSB's of radio altimeter data, LSB first) (2 bits)
   - e. Data bits (32 bits, LSB first)
     - 1) Radio altimeter data (8 bits)
     - 2) Air temperature data (8 bits)
     - 3) Air pressure data (8 bits)
     - 4) Pressure temperature data (8 bits)
4. Transmitter power-down

```
;     TWERLE FORMAT TRANSMITTER SUBROUTINE      ;
;                                               ;
; THIS PROGRAM CONTROLS THE TRANSMISSION OF 34  ;
;  BITS OF MANCHESTER ENCODED DATA IN TWERLE    ;
;    FORMAT USING AN 8080A MICROPROCESSOR       ;
;       WITH A 1.6666 MICROSECOND STATE         ;
;         TIME.  ALL TIMING FUNCTIONS           ;
;             ARE ACCOMPLISHED IN               ;
;                   SOFTWARE                    ;


                        DATA CONSTANT DEFINITIONS

020000                  TWD     = 20000 ; TWERLE DATA BLOCK AT 4K POINT
000000                  XMIT    = 0     ; OUTPUT PORT ASSIGNMENTS
000001                  XCNTR   = 1     ;
000252                  BSYNC   = 252   ; BIT SYNC PATTERN
000153                  FSYNC   = 153   ; FRAME SYNC PATTERN
000106                  TWPO    = 106   ; TWERLE POWER ON CONSTANT
000146                  TWCC    = 146   ; TWERLE CLEAR CARRIER CONSTANT
000142                  TWDT    = 142   ; TWERLE DATA TRANSMISSION CONSTANT
000004                  TO      = 4     ; TRANSMITTER OFF CONSTANT (POWER OFF)
000000                  ADRS    = 0     ; TWERLE PDCP ADDRESS INPUT PORT ASSIGNMENT
000001                  ADRS1   = 1     ; TWERLE PDCP ADDRESS - 2MSB INPUT PORT


000000                  ORG     3244    ; START PROGRAM AT 1700.


003244          TWXMIT:                 ; STARTING POINT OF TWERLE TRANSMITTER ROUTINE
003244                  MVI     A,TWPO  ; APPLY TRANSMITTER POWER IN TWERLE MODE
003244  076 106
003246                  OUT     XCNTR   ; WITH RF TRANSMISSION INHIBITED
003246  323 001
003250                  LXI     H,24966. ; PROVIDE 1 SECOND WARM-UP DELAY
003250  041 206 141
003253                  CALL    DELAY1
003253  315 040 007
003256                  MVI     A,TWCC  ; ENABLE TRANSMISSION WITHOUT MODULATION
003256  076 146
003260                  OUT     XCNTR   ; DO IT!
003260  323 001
```

Program 4.4.3.1(1) TWERLE Format Transmitter Subroutine.

```
                        ;                CLEAR CARRIER SUBROUTINE                ;
                        ;                                                        ;
                        ;   CLEAR CARRIER TRANSMISSION TIME VARIES FROM          ;
                        ;   0.31952 SEC. TO 0.36048 SEC.  DEPENDING ON THE       ;
                        ;   PLATFORM ADDRESS.   THIS PREVENTS PLATFORM TO        ;
                        ;     PLATFORM INTERFERENCE, AS REQUIRED BY THE          ;
                        ;     RAMS SYSTEM ABOARD THE NIMBUS F SATELLITE          ;

003262
003262  001  374  037      LXI   B,TWD-4  ; ESTABLISH BC AS AN ADDRESS REGISTER
                                          ; POINTING 4 LOCATIONS BEFORE THE TWERLE
                                          ; RAM DATA BLOCK
003265
003265  076  252          MVI   A,BSYNC  ; BIT SYNC BYTE ----> ACCUMULATOR
003267
003267  002              STAX  B        ; BSYNC ----> <TWD-4>
003270
003270  076  153          MVI   A,FSYNC  ; FRAME SYNC BYTE ----> ACCUMULATOR
003272
003272  003              INX   B        ; POINT BC TO TWD-3
003273
003273  002              STAX  B        ; STORE 1ST EIGHT FRAME SYNC BITS IN <TWD-3>
003274
003274  041  046  037      LXI   H,7974.  ; DELAY TO ESTABLISH MINIMUM CLEAR CARRIER
003277
003277  315  042  007      CALL  DELAY2   ; TRANSMISSION TIME OF 0.31952 SECONDS
003302                     DLY4
003302  000
003303
003303  333  000          IN    ADRS     ; GET 8 LEAST SIGNIFICANT BITS OF TWERLE PDCP ADDRESS
003305
003305  157              MOV   L,A      ; 8 LSB'S OF PLATFORM ADDRESS ----> L
003306
003306  137              MOV   E,A      ; 8 LSB'S OF PLATFORM ADDRESS ----> E
003307
003307  333  001          IN    ADRS+1   ; GET TWO MSB'S OF PLATFORM ADDRESS
003311
003311  346  003          ANI   3        ; PICK OFF BOTTOM TWO BITS ONLY
003313
003313  147              MOV   H,A      ; 2 MSB'S OF PLATFORM ADDRESS ----> H
003314
003314  127              MOV   D,A      ; 2 MSB'S OF PLATFORM ADDRESS ----> D
003315
003315  051              DAD   H        ; DOUBLE PRECISION 4 BIT SHIFT LEFT OF HL
003316
003316  051              DAD   H
003317
003317  051              DAD   H
003320
003320  051              DAD   H
003321
003321  003              INX   B        ; POINT BC TO TWD-2
003322
003322                   MOV   A,L      ; L ----> A AND STORE THE LAST FOUR FRAME SYNC
```

Program 4.4.3.1(1) (continued)

```
003322  175
003323
003323  002
                                        STAX    B       ; BITS AND THE FIRST FOUR PLATFORM ADDRESS BITS
                                                        ; IN <TWD-2>
003324                                  INX     B       ; POINT BC TO TWD-1
003324  003
003325                                  MOV     A,H     ; H ----->A AND STORE THE 6 MSB'S OF THE PLATFORM
003325  174
003326                                  STAX    B       ; ADDRESS IN <TWD-1>
003326  002
003327                                  XCHG            ; RETURN VALUE OF PLATFORM ADDRESS TO HL
003327  353
003330                                  INX     H       ; INCREMENT HL SO HL .NE. 0
003330  043
003331                                  CALL    DELAY   ; ADD DELAY OF 40*(HL) MICROSECONDS.  THE TOTAL
003331  315     044     007
                                                        ; CLEAR CARRIER TRANSMISSION IS 0.31952 TO
                                                        ; 0.36048 SECONDS DEPENDING ON THE PLATFORM ADDRESS
003334                                  LXI     B,3410  ; 7 ----> B    AND    8 ----> C
003334  001     010     007
003337                                  LXI     H,TWD-4 ; ESTABLISH HL AS AN ADDRESS REGISTER POINTING TO
003337  041     374     037
                                                        ; THE FIRST BYTE TO BE TRANSMITTED FROM RAM
003342                                  MVI     A,TWDT  ; PREPARE TO TRANSMIT THE FIRST BIT OF BIT
003342  076     142
003344                                  OUT     XMIT    ; SYNC BY PLACING A '1' IN THE TRANSMITTER
003344  323     000
003346                                  OUT     XCNTR   ; DATA BIT LATCH AND THEN ENABLE TRANSMITTER
003346  323     001
                                                        ; MODULATION
003350                                  MOV     A,M     ; BSYNC BYTE ----> ACCUMULATOR
003350  176
003351                                  LXI     D,197.  ; SET UP DELAY TO ESTABLISH
003351  021     305     000
                                                        ; 100 BPS TRANSMISSION RATE
003354                                  JMP     TW2-1
003354  303     365     006
```


Program 4.4.3.1(1) (continued)

```
                        ; THIS SUBROUTINE CONTROLS TRANSMISSION OF THE REMAINING 7 BITS ;
                        ;  OF BIT SYNC, THE 12 BITS OF FRAME SYNC, THE 10 BIT PLATFORM  ;
                        ;    ADDRESS, AND THE 34 DATA BITS FROM RAM BEGINNING AT RAM    ;
                        ;                    ADDRESS TWD-4                              ;

003357                  TW1:
003357                          INX     H       ; POINT HL AT THE NEXT DATA BYTE
003357  043
003360                          MVI     B,10    ; BIT COUNT=8 --> B
003360  006    010
003362                          MOV     A,M     ; GET THE DATA BYTE INTO THE ACCUMULATOR
003362  176
003363                          MVI     E,194.  ; DELAY TO PRODUCE 100 BPS DATA RATE
003363  036    302
003365                          DLY4
003365  000
003366                  TW2:    DCR     E
003366  035
003367                          JNZ     TW2
003367  302    366    006
003372                          CMA             ; COMPLEMENT ACCUMULATOR
003372  057
003373                          OUT     XMIT    ; OUTPUT FIRST PHASE OF CURRENT BIT
003373  323    000
003375                          CMA             ; COMPLEMENT ACCUMULATOR
003375  057
003376                          MVI     E,198.  ; DELAY TO PRODUCE 100 BPS DATA RATE
003376  036    306
003400                  TW3:    DCR     E
003400  035
003401                          JNZ     TW3
003401  302    000    007
003404                          DLY9
003404  000
003405  177
003406                          OUT     XMIT    ; TRANSMIT THE SECOND PHASE OF THE CURRENT BIT
003406  323    000
003410                          RRC             ; ROTATE THE DB RIGHT ONCE TO GET NEXT BIT
003410  017
003411                          DCR     B       ; IF ALL BITS OF THE CURRENT BYTE HAVE NOT
003411  005
003412                          DLY5            ; BEEN TRANSMITTED, PREPARE TO JUMP TO TW2
003412  177
003413                          MVI     E,197.  ; SET UP DELAY
003413  036    305
003415                          JNZ     TW2     ; JUMP IF THE LAST BIT OF THIS BYTE HAS NOT
003415  302    366    006
                                                ; BEEN TRANSMITTED
003420                          DCR     C       ; IF THE LAST DATA BYTE HAS NOT BEEN TRANSMITTED,
003420  015
003421                          JNZ     TW1     ; GO GET THE NEXT BYTE
003421  302    357    006
003424                          DLY5            ; OTHERWISE DELAY UNTIL THE END OF THE SECOND
003424  177
```

Program 4.4.3.1(1) (continued)

```
003425                                  MVI     E,195.  ; PHASE OF THE LAST BIT
003425  036     303
003427                          TW4:    DCR     E
003427  035
003430                                  JNZ     TW4
003430  302     027     007
003433                                  MVI     A,TO    ; PREPARE TO SHUT DOWN TRANSMITTER RF
003433  076     004
003435                                  OUT     XCNTR   ; DO IT!
003435  323     001
003437                                  RET             ; RETURN TO EXECUTIVE
003437  311

                                ; SUBROUTINES FOR DELAY ;

003440                          DELAY1: DLY21
003440  305     301
003442                          DELAY2: DLY21
003442  305     301
003444                          DELAY:  DCX     H       ; LONG DELAY SUBROUTINE DECREMENTS
003444  053
003445                                  MOV     A,L     ; REGISTER PAIR HL UNTIL ZEROED. CREATES A DELAY
003445  175
003446                                  ORA     H       ; OF 27+24*(HL) STATES COUNTING THE CALL AND RETURN
003446  264
003447                                  JNZ     DELAY   ; INSTRUCTIONS. DELAY1 AND DELAY2 OPTIONS
003447  302     044     007
003452                                  RET             ; ADD UP TO 42 STATES ADDITIONAL DELAY.
003452  311

                                        END
```


Program 4.4.3.1(1) (continued)

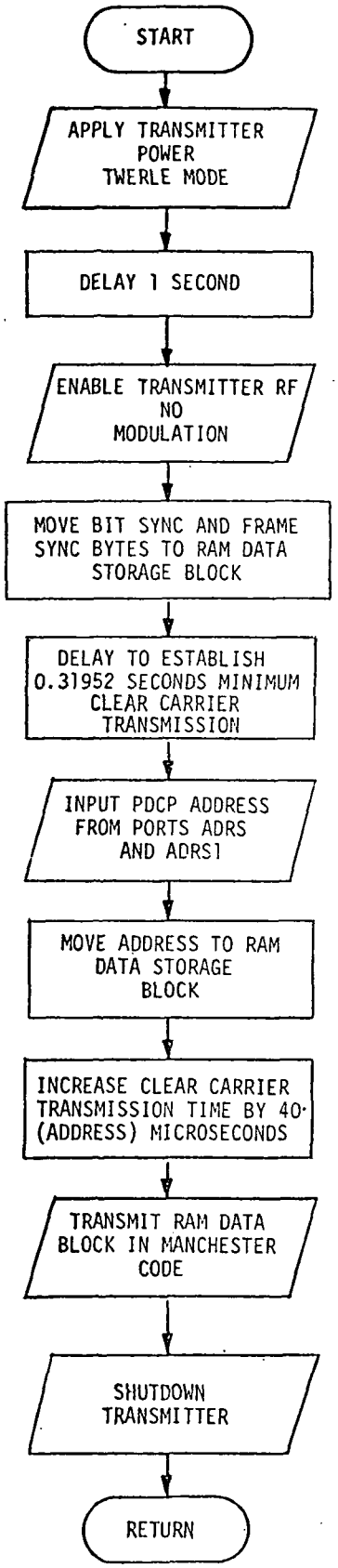


Figure 4.4.3.1(1) Simplified Flow Chart for Subroutine TWXMIT.

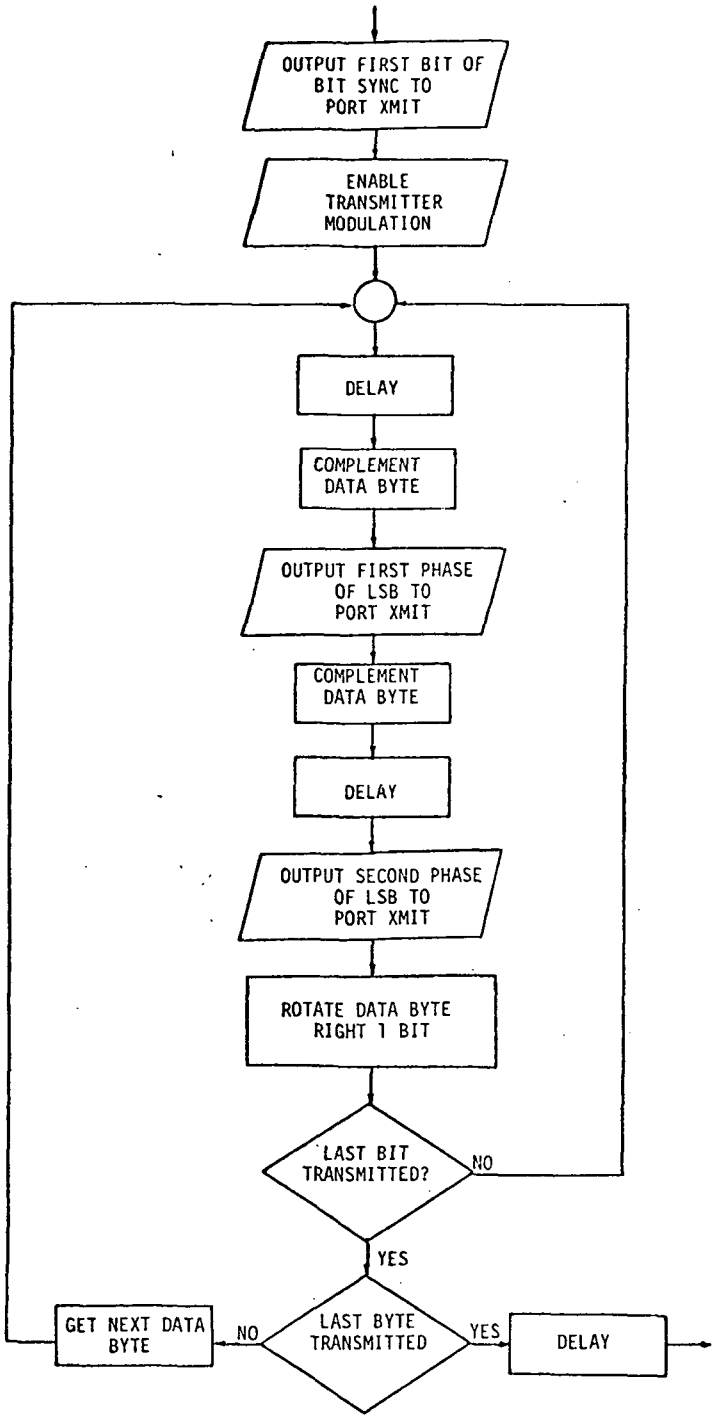


Figure 4.4.3.1(2) Detailed Flow Chart for the Data Transmission Block of Subroutine TWXMIT.

Each TWERLE DCP is assigned a unique ten-bit address code which is normally plug-wired [7]. The TWXMIT subroutine assumes a similar arrangement in which the PDCP address is plug-wired on two bits of input port ADRS1 and eight bits of input port ADRS. These input ports are sampled during the clear carrier portion of each transmission, and the plug-wired address is temporarily stored in the RAM data block. The bit and frame synchronization codes are also moved to the RAM data storage block temporarily so that the data can be transmitted from sequential memory locations. Table 4.4.3.1(2) illustrates the format of the RAM data block.

TABLE 4.4.3.1(2)

FORMAT OF THE TWERLE RAM DATA BLOCK

Memory Address	Data Byte							
	MSB							LSB
TWD-4	1	0	1	0	1	0	1	0
TWD-3	0	1	1	0	1	0	1	1
TWD-2	X_3	X_2	X_1	X_0	0	0	0	0
TWD-1	A_9	A_8	X_9	X_8	X_7	X_6	X_5	X_4
TWD	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
TWD+1	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0
TWD+2	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
TWD+3	Pt_7	Pt_6	Pt_5	Pt_4	Pt_3	Pt_2	Pt_1	Pt_0

KEY: X - TWERLE PDCP Address Plug-Wired to Ports ADRS and ADRS+1

A - Altimeter Data

T - Air Temperature Data

P - Pressure Data

Pt - Pressure Temperature Data

TWERLE DCP's transmit to the RAM's system aboard the Nimbus-F satellite. This system relies on time and frequency division multiplexing of the data from the various DCP's to prevent transmission interference [8]. In present TWERLE DCP designs, time-division multiplexing depends upon establishing a random variation in the length of the clear carrier transmission from each DCP. An average clear carrier transmission time of 0.34 seconds with nominal range of 0.32 to 0.36 seconds is desired. Currently, an imprecise one-shot circuit which must be manually adjusted for proper timing is used to control the clear carrier transmission time [7] and prevent two systems from interferring with each other for an excessive period. The TWXMIT subroutine listed as Program 4.4.3.1(1) makes a substantial improvement over this system. A software delay routine establishes a minimum clear carrier transmission time of 0.31952 seconds which may be increased to a maximum of 0.36048 seconds according to the formula

$$T = 319.52 + 40A,$$

where T = clear carrier transmission time in microseconds, and A = decimal equivalent to PDCP TWERLE address. Since each TWERLE PDCP is assigned a unique address, this technique guarantees that two systems will not remain in time sequence and interfere with each other. In addition, the transmission interval for each TWERLE PDCP can be computed from the address assigned to that system. Any variation from the assigned transmission interval would provide an early warning of possible system malfunction. Also, the need for manual adjustment of the transmission interval is eliminated.

The TWERLE format transmitter routine uses a general-purpose, long-delay routine which is 11 bytes long. Only 123 additional bytes of ROM are required to store the program. At present 100-piece CMOS 512 byte ROM prices, the incremental component cost of providing the TWXMIT subroutine as part of a standard PDCP ROM is approximately $9.85. In addition, this subroutine utilizes 12 bytes of RAM for data and stack storage, 10 input bits to specify the platform address, and 8 output bits to provide data and control to the transmitter.

4.4.3.2 GOES Format Transmitter Routine - Specifications for the GOES data transmission sequence [6] are presented in Table 4.4.3.2(1). Program 4.4.3.2(1) is an assembly language listing of the GOES format transmitter routine written for the UT PDCP. This subroutine provides all necessary data formatting and encoding as well as complete transmitter control. In contrast to the TWERLE format transmitter subroutine, Program 4.4.3.2(1) assumes that the unique 31-bit address code assigned to each GOES PDCP is either stored in PROM or plug-wired to a memory-oriented input port. Both ASCII and Manchester data encoding are accomplished in real time during transmission to minimize data storage requirements.

A flow chart for the basic GOES format transmitter subroutine is illustrated in Figure 4.4.3.2(1), and a flow chart for the BIPHS subroutine called by the GOES subroutine is shown in Figure 4.4.3.2(2). Since the amount of GOES data transmitted is variable, the first byte of the RAM data storage block is used to indicate the current number of data bytes. Twelve additional bytes of RAM are required for stack operations. The GOES transmitter program, including subroutine BIPHS and the table area required to store preamble data, occupies 225 bytes of ROM. This program also requires the same 11-byte, general-purpose long-delay subroutine used by the TWERLE format transmitter routine. Primarily because of the ASCII encoding requirement, the GOES format transmitter subroutine is significantly longer than either the TWERLE, TIROS-N, or Landsat transmitter programs. Nevertheless, the GOES transmitter routine would occupy only one-fourth of an 8K-bit ROM. The incremental component cost of providing this subroutine in a standard PDCP ROM is approximately $18.02.

4.4.3.3 TIROS-N Format Transmitter Routine - Program 4.4.3.3(1) is an assembly language listing of the TIROS-N format transmitter subroutine written for the UT PDCP. This program is based on the preliminary TIROS-N DCP specifications [9] presented in Table 4.4.3.3(1). A flow chart for Program 4.4.3.3(1) is illustrated in Figure 4.4.3.3(1). The TIROS-N transmitter subroutine provides real-time Manchester encoding of the preamble and the 32 data bits. Data is transmitted at a rate of 400 bits per second using the standard transmitter control signals

TABLE 4.4.3.2(1)

GOES DATA TRANSMISSION SEQUENCE SPECIFICATIONS

Transmission Interval:	Selectable up to 24.75 hours in 0.25 hour increments
Transmission Rate:	100 bits per second
Coding:	Preamble - Manchester encoded with a 0 → 1 transition representing a 1 Data and EOT characters - ASCII encoded then Manchester encoded as above

Transmission Sequence:

1. Transmitter power-up followed by a 1-second warm-up delay
2. Clear carrier transmission for a minimum of 5 seconds
3. Preamble transmission
   a. Bit synchronization clock (250 bits minimum)
   b. Frame synchronization code (1000100110101111) (15 bits)
   c. Address code (assigned to user) (31 bits)
4. Data transmission (up to 2000 bits)
5. EOT code transmission - three ASCII end-of-transmission characters (33 bits)
6. Transmitter power-down

```
                        ;                GOES FORMAT TRANSMITTER PROGRAM                ;
                        ;   THIS PROGRAM CONTROLS TRANSMISSION OF UP TO 2000 BITS OF    ;
                        ;     MANCHESTER ENCODED DATA IN GOES FORMAT USING AN 8080A     ;
                        ;     MICROPROCESSOR WITH A 1.66666 MICROSECOND STATE TIME      ;
                        ;                                                               ;
                        ;      ALL TIMING FUNCTIONS ARE ACCOMPLISHED IN SOFTWARE        ;
000000                          ORG     1000    ; START PROGRAM AT 1/4 K POINT

        020000          GDATA=20000     ; GOES DATA BLOCK AT 4K POINT
        000000          XMIT=0          ; OUTPUT PORT ASSIGNMENT CONSTANTS
        000001          XCNTR=1
        000124          GTPO=124        ; GOES TRANSMITTER POWER ON CONSTANT
        000164          GCC=164         ; GOES CLEAR CARRIER CONSTANT
        000160          GDT=160         ; GOES DATA TRANSMISSION CONSTANT
        000004          TO=004          ; TRANSMITTER OFF CONSTANT
                        ;
001000                  GXMIT:
001000                          MVI     A,GTPO  ; APPLY TRANSMITTER POWER IN GOES MODE
001000  076     124
                                                ; WITH TRANSMISSION INHIBITED
001002                          OUT     XCNTR
001002  323     001
001004                          LXI     H,24996.; PROVIDE 1-SECOND WARM-UP DELAY
001004  041     244     141
001007                          CALL    DELAY1
001007  315     271     002
001012                          MVI     A,GCC   ; ENABLE TRANSMISSION WITHOUT MODULATION
001012  076     164
001014                          OUT     XCNTR   ; DO IT
001014  323     001
001016                          MVI     B,4     ; PROVIDE 5-SECOND DELAY FOR CLEAR CARRIER
001016  006     004
001020                  GX:     LXI     H,31247.; FOLLOWING THIS DELAY, THE FIRST OF
001020  041     017     172
001023                          CALL    DELAY1  ; 251 CLOCK BITS IS TRANSMITTED IN
001023  315     271     002
001026                          DCR     B       ; MANCHESTER CODE WITH A "ONE" REPRESENTED
001026  005
                                                ; BY A 0-->1 TRANSITION
001027                          JNZ     GX      ;
001027  302     020     002
001032                          DLY4            ;
001032  000
001033                          MVI     A,GDT   ; PREPARE TO TRANSMIT 1ST PHASE OF CLOCK BIT 1
001033  076     160
001035                          OUT     XMIT    ; BY PLACING A '0' IN TRANSMITTER DATA BIT LATCH
001035  323     000
                                                ; AND ENABLING TRANSMITTER MODULATION
001037                          OUT     XCNTR   ; DO IT
001037  323     001
001041                          LXI     H,122.  ; INSERT DELAY TO CREAT 100 BPS TRANS RATE
001041  041     172     000
001044                          CALL    DELAY2
001044  315     273     002
```

Program 4.4.3.2(1) GOES [illegible]

```
001047                                  CMA
001047  057
001050                                  OUT     XMIT
001050  323     000
001052                                  DAD     H       ; DELAY 10 STATES
001052  051
001053                                  DAD     H       ; DELAY 10 STATES
001053  051
001054                                  DAD     H       ; DELAY 10 STATES
001054  051
001055                                  LDA     GDATA   ; FIND THE NUMBER OF DATA BYTES TO BE
001055  072     000     040
                                        ; TRANSMITTED AND STORE THE NUMBER
001060                                  MOV     C,A     ; IN REGISTER C
001060  117
001061                                  LXI     H,PAD-1 ; ESTABLISH HL AS AN ADDRESS REGISTER POINTING
001061  041     303     002
                                        ; TO THE MEMORY LOCATION IMMEDIATELY PRE-
                                        ; CEEDING THE GOES ROM DATA STORAGE AREA
001064                                  MVI     D,45
001064  026     045
001066                          GX1:    DLY5
001066  177
001067                                  INX     H       ; INCREMENT HL TO POINT TO NEXT DB MEMORY LOCATION
001067  043
001070                          GX2:    MVI     B,10    ; B SPECIFIES THE NUMBER OF BITS TO BE TRANSMITTED
001070  006     010
                                        ; FROM THE CURRENT BYTE
001072                                  MOV     A,M     ; MOVE THE NEXT DB ---> ACCUMULATOR
001072  176
001073                                  PUSH    H
001073  345
001074                                  MOV     H,M     ; DELAY 7 STATES
001074  146
001075                                  MVI     H,187.
001075  046     273
001077                                  DLY4
001077  000
001100                                  CALL    BIPHS   ; CALL BIPHASE ROUTINE WHICH CONTROLS
001100  315     316     002
                                        ; TRANSMISSION OF THE CURRENT DATA WORD
                                        ; IN MANCHESTER CODE AT 100 BPS
001103                                  POP     H
001103  341
001104                                  MVI     A,7
001104  076     007
001106                                  DCR     D       ; IF THE FINAL POOP GOES ADDRESS BYTE
001106  025
001107                                  JZ      GX3     ; HAS BEEN TRANSMITTED, GOTO GOES DATA
001107  312     121     002
                                        ; TRANSMISSION ROUTINE
001112                                  CMP     D       ; IF THE LAST FULL CLOCK BYTE HAS BEEN
001112  272
001113                                  JC      GX1     ; TRANSMITTED, JUMP TO GX1 AND INCREMENT HL
001113  332     066     002
001116                                  JMP     GX2     ; OTHERWISE GOTO GX2 AND CONTINUE TRANSMITTING
001116  303     070     002
```

Program 4.4.3 (Cont'd)

```
001121                                  ;  CLOCK BITS
001121          176             GX3:    MOV     A,M     ;  DELAY 7 STATES AND THEN EXTABLISH HL AS
001122                                  LXI     H,GDATA ;  AN ADDRESS REGISTER POINTING TO THE GOES
001122          041     000     040
                                        ;  RAM DATA STORAGE AREA
001125                          GX4:    INX     H       ;  INCREMENT HL TO POINT TO THE NEXT DB MEMORY LOCATION
001125          043
001126                                  MOV     A,M     ;  MOVE THE NEXT DB --> ACCUMULATOR
001126          176
001127                                  RLC             ;  ROTATE THE DB LEFT ONE BIT
001127          007
001130                                  MVI     D,2     ;  D IS A COUNTER WHICH FLAGS WHEN BOTH NIBBLES OF
001130          026     002
                                        ;  THE CURRENT DB HAVE BEEN TRANSMITTED
001132                          GX5:    MVI     B,10    ;  B SPECIFIES THE NUMBER OF BITS TO BE TRANSMITTED
001132          006     010
                                        ;  FROM THE CURRENT BYTE
001134                                  MVI     E,7     ;  PREPARE TO CONVERT THE CURRENT NIBBLE TO ASCII.  E IS
001134          036     007
                                        ;  THE PARITY FLAG
001136                                  ANI     36
001136          346     036
001140                                  ORI     140
001140          366     140
001142                                  JPO     GX10    ;  CHECK PARITY AND DECREMENT E ONLY IF PARITY IS EVEN
001142          342     266     002
001145                                  DCR     E       ;  EVEN PARITY IS SENT-TO CHANGE TO ODD, CHANGE JPO
001145          035
                                        ;  INSTRUCTION TO JPE
001146                                  DLY5
001146          177
001147                          GX6:    PUSH    H       ;  SAVE HL ON STACK
001147          345
001150                                  MVI     H,183.  ;  ESTABLISH LENGTH OF INITIAL BIPHASE
001150          046     267
001152                                  DLY13           ;  DELAY
001152          000
001153          000
001154          177
001155                                  CALL    BIPHS   ;  CALL BIPHASE SUBROUTINE TO TRANSMIT THE FIRST 8 BITS
001155          315     316     002
                                        ;  OF THE CURRENT NIBBLES ASCII CODE
001160                                  MVI     B,3     ;  B SPECIFIES THE NUMBER TO BE TRANSMITTED
001160          006     003
                                        ;  FROM THE CURRENT BYTE
001162                                  MOV     A,E     ;  MOVE LAST 3 BITS OF CURRENT ASCII WORD TO ACCULATOR
001162          173
001163                                  MVI     H,191.
001163          046     277
001165                                  CALL    BIPHS   ;  CALL BIPHASE TO TRANSMIT THE REMAINING 3 BITS
001165          315     316     002
                                        ;  OF THE CURRENT NIBBLE'S ASCII CODE
001170                                  POP     H       ;  RESTORE HL FROM THE STACK
001170          341
001171                                  DCR     D       ;  IF BOTH NIBBLES OF THE CURRENT BYTE HAVE BEEN
001171          025
```

Program 4.4.3.2(1) (continued)

```
001172                                  JZ      GX7     ; CONVERTED TO ASCII, JUMP TO GX7
001172  312     210     002
001175                                  MOV     A,M     ; IF NOT, MOVE THE ORIGINAL BYTE INTO ACCUMULATOR
001175  176
001176                                  RRC             ; AND ROTATE RIGHT THREE TIMES
001176  017
001177                                  RRC
001177  017
001200                                  RRC
001200  017
001201                                  DLY16
001201  000     000
001203  000     000
001205                                  JMP     GX5     ; RETURN TO GX5 TO TRANSMIT ASCII CODE OF SECOND NIBBLE
001205  303     132     002
001210                          GX7:    DLY5
001210  177
001211                                  DCR     C       ; DECREMENT THE DATA BYTE COUNTER
001211  015
001212                                  JNZ     GX4     ; JUMP TO GX4 TO TRANSMIT THE NEXT DATA BYTE
001212  302     125     002
                                        ; IF C IS NOT ZERO
001215                                  MVI     C,3     ; ELSE RESET C TO 3
001215  016     003
001217                          GX8:    MVI     A,10    ; 8 ---> ACCUMULATOR
001217  076     010
001221                                  MOV     B,A     ; 8 ---> B
001221  107
001222                                  LXI     H,48128. ; DELAY 3 STATES AND ESTABLISH LENGTH OF INITIAL
001222  041     000     274
                                        ; BIPHASE DELAY BY LOADING 188. TO H
001225                                  CALL    BIPHS   ; CALL BIPHASE TO TRANSMIT THE FIRST 8 BITS OF
001225  315     316     002
                                        ; AN END OF TRANSMISSION (EOT) CHARACTER
001230                                  MVI     A,6
001230  076     006
001232                                  MVI     B,3
001232  006     003
001234                                  MVI     H,191.  ; ESTABLISH LENGTH OF INITIAL BIPHASE DELAY
001234  046     277
001236                                  DLY8
001236  000     000
001240                                  CALL    BIPHS   ; CALL BIPHASE TO TRANSMIT THE REMAINING 3
001240  315     316     002
                                        ; BITS OF AN 'EOT' CHARACTER
001243                                  MOV     H,M     ; DELAY 7 STATES
001243  146
001244                                  DAD     H       ; DELAY 10 STATES
001244  051
001245                                  DAD     H       ; DELAY 10 STATES
001245  051
001246                                  DAD     H       ; DELAY 10 STATES
001246  051
001247                                  DCR     C       ; IF THE LAST (3RD) 'EOT' CHARACTER HAS NOT BEEN
001247  015
001250                                  JNZ     GX8     ; TRANSMITTED, JUMP TO GX8 AND TRANSMIT THE NEXT CHAR.
001250  302     217     002
```

Program 4.4.3.2(1) (continued)

```
001253                                  MVI     A,193.  ; ELSE ADD DELAY
001253  076     301
001255                          GX9:    DCR     A       ; DELAY
001255  075
001256                                  JNZ     GX9
001256  302     255     002
001261                                  MVI     A,TO    ; SHUT DOWN TRANSMITTER
001261  076     004
001263                                  OUT     XCNTR   ; DO IT!
001263  323     001
001265                                  RET             ; RETURN TO EXECUTIVE
001265  311

001266                          GX10:   JMP     GX6     ; DELAY
001266  303     147     002
                                        ;
                                        ;
                                        ;
001271                          DELAY1: DLY21
001271  305     301
001273                          DELAY2: DLY21
001273  305     301
001275                          DELAY:
001275                                  DCX     H       ; LONG DELAY SUBROUTINE.  DECREMENTS HL UNTIL
001275  053
001276                                  MOV     A,L     ; ZEROED.  CREATES A DELAY OF 27+24*<HL> STATES
001276  175
001277                                  ORA     H       ; COUNTING THE CALL AND RETURN INSTRUCTIONS
001277  264
001300                                  JNZ     DELAY
001300  302     275     002
001303                                  RET
001303  311

001304                          PAD:    DB      377     ; THIS BLOCK IS DATA STORAGE FOR A CLOCK WORD,
001304  377
                                        ; A 15-BIT SYNC WORD AND A 31-BIT PDCP GOES ADDRESS
001305                                  DB      107
001305  107
001306                                  DB      326     ; THE 31-BIT GOES ADDRESS SHOULD BE STORED AS SHOWN BELOW:
001306  326
001307                                  DB      001     ; STORE BITS A6-A0 AS THE MSB THROUGH THE NEXT TO LSB
001307  001
                                        ; OF THIS WORD.  THE LSB MUST BE 1.
001310                                  DB      000     ; STORE BITS A14-A7 AS THE MSB THROUGH THE LSB OF THIS WOI
001310  000
001311                                  DB      000     ; STORE BITS A22-A15 AS THE MSB THROUGH THE LSB OF THIS W
001311  000
001312                                  DB      000     ; STORE BITS A30-A23 AS THE MSB THROUGH THE LSB OF THIS W
001312  000
                                        ; A0 IS THE LSB OF THE PDCP ADDRESS AND A30 IS THE
                                        ; MSB OF THE PDCP ADDRESS.  THE ADDRESS CURRENTLY STORED
                                        ; IS ZERO.
```

Program 4.4.3.2(1) (continued)

```
                        ; THIS SUBROUTINE CONTROLS TRANSMISSION OF THE BYTE CURRENTLY ;
                        ;  STORED IN THE ACCUMULATOR.  THE DATA IS MANCHESTER ENCODED  ;
                        ;                    AS IT IS TRANSMITTED                      ;

001313                  BI2:    DLY4
001313  000
001314                          MVI     H,194.
001314  046  302
001316                  BIPHS:
001316                          DLY36
001316  343  343
001320                  BI1:    DCR     H       ; DELAY TO CREATE 100 BPS TRANSMISSION RATE
001320  045
001321                          JNZ     BI1
001321  302  320  002
001324                          JMP     BIOUT
001324  303  327  002

001327                  BIOUT:  CMA             ; COMPLEMENT THE DATA BYTE
001327  057
001330                          OUT     XMIT    ; TRANSMIT THE 1ST PHASE OF THE CURRENT LSB
001330  323  000
001332                          CMA             ; COMPLEMENT THE DATA BYTE AGAIN
001332  057
001333                          LXI     H,121.
001333  041  171  000
001336                          PUSH    PSW
001336  365
001337                          CALL    DELAY   ; DELAY TO CREATE 100 BPS TRANSMISSION RATE
001337  315  275  002
001342                          POP     PSW
001342  361
001343                          OUT     XMIT    ; TRANSMIT THE 2ND PHASE OF THE CURRENT LSB
001343  323  000
001345                          RRC             ; ROTATE THE DB RIGHT 1 BIT POSITION
001345  017
001346                          DCR     B       ; IF BIT '8' OF THE CURRENT DB HAS NOT BEEN
001346  005
001347                          JNZ     BI2     ; TRANSMITTED, JUMP TO BI2 AND INSERT DELAY
001347  302  313  002
001352                          RET             ; OTHERWISE RETURN TO THE MAIN GXMIT PROGRAM
001352  311


001353                          END
```


Program 4.4.3.2(1) (continued)

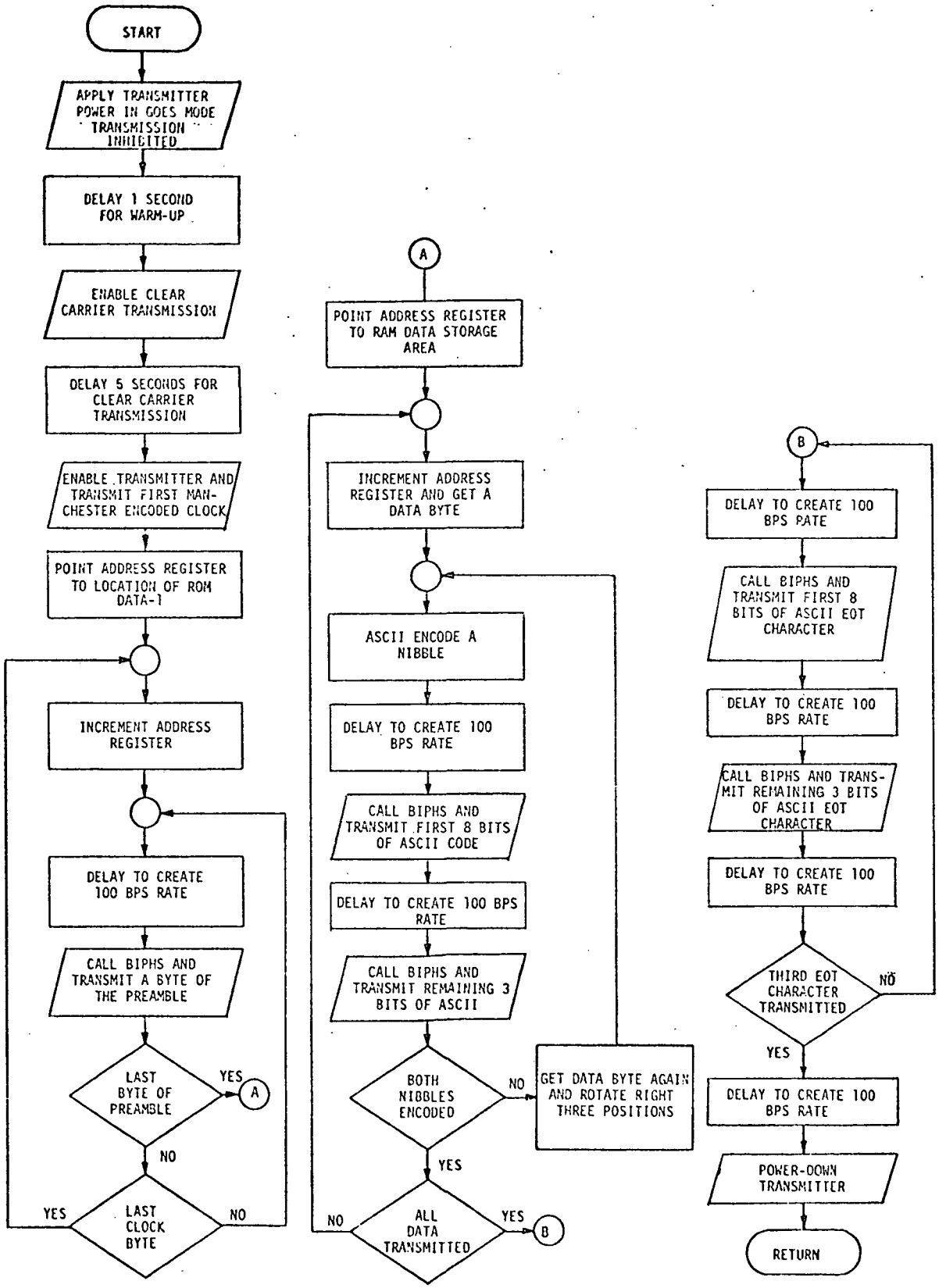


Figure 4.4.3.2(1) Flow Chart for GOES Transmitter Subroutine.

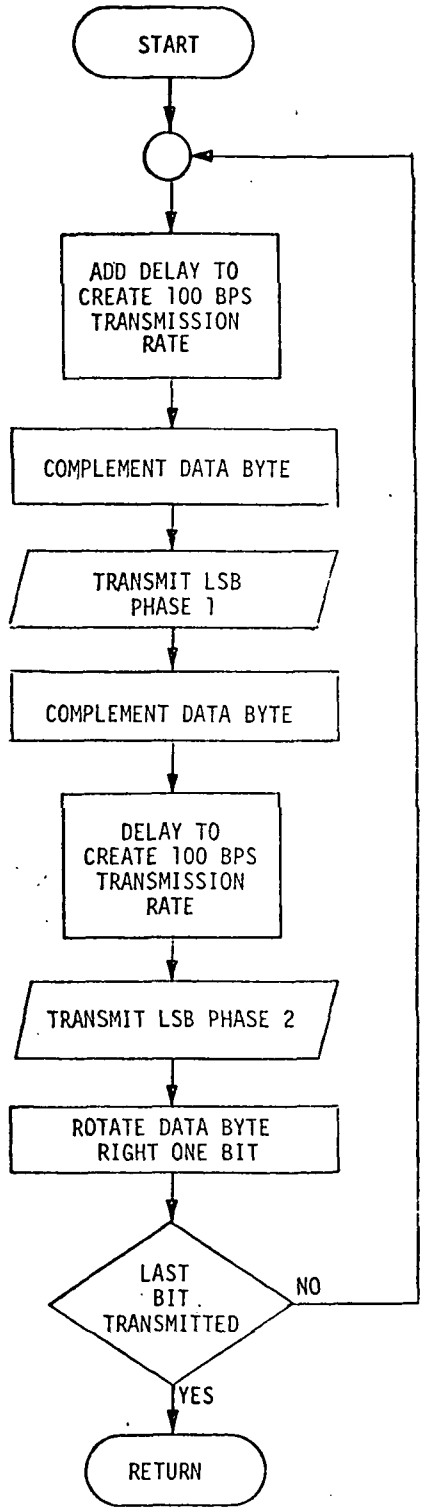


Figure 4.4.3.2(2) Flow Chart for Subroutine BIPHS.

```
                                TIROS-N FORMAT ---- TRANSMITTER SUBROUTINE

                        ; THIS PROGRAM CONTROLS THE TRANSMISSION OF 32 BITS OF MANCHESTER ;
                        ;  ENCODED DATA IN TIROS-N FORMAT USING AN 8080A MICROPROCESSOR   ;
                        ;  WITH A 1.66666 MICROSECOND STATE TIME.  ALL TIMING FUNCTIONS   ;
                        ;  ARE ACCOMPLISHED IN SOFTWARE.                                  ;

                           DATA CONSTANT DEFINITIONS:

020000                          TND=20000       ; TIROS-N DATA BLOCK AT 4K POINT

000000                          XMIT=0          ; OUTPUT PORT ASSIGNMENTS
000001                          XCNTR=1

000304                          TTPO=304        ; TIROS TRANSMITTER POWER ON CONSTANT
000344                          TCC=344         ; TIROS CLEAR CARRIER CONSTANT
000340                          TDT=340         ; TIROS DATA TRANSMISSION CONSTANT


000000                          ORG     3000    ; START PROGRAM AT 1.5K POINT

003000                  TNXMIT:                 ; STARTING POINT OF TIROS TRANSMITTER
                                                ; ROUTINE.

003000                          MVI     A,TTPO  ; APPLY TRANSMITTER POWER IN TIROS-N MODE
003000  076     304
                                                ; WITH TRANSMISSION INHIBITED
003002                          OUT     XCNTR   ; DO IT
003002  323     001
003004                          LXI     H,24997. ; PROVIDE ONE-SECOND WARM-UP DELAY
003004  041     245     141
003007                          CALL    DELAY
003007  315     167     006
003012                          DLY10
003012  177     177
003014                          MVI     A,TCC   ; ENABLE TIROS CLEAR CARRIER TRANSMISSION
003014  076     344
003016                          OUT     XCNTR   ; DO IT
003016  323     001
003020                          LXI     H,3997. ; PROVIDE 160 MILLISECOND DELAY
003020  041     235     017
003023                          CALL    DELAY   ; FOLLOWING THIS DELAY, THE FIRST BIT
003023  315     167     006
                                                ; OF THE BIT SYNCHRONIZATION PATTERN
003026                          MVI     A,TDT   ; 111111111111111 IS TRANSMITTED
003026  076     340
003030                          OUT     XMIT    ; IN MANCHESTER CODE WITH A
003030  323     000
                                                ; ONE REPRESENTED BY A 0-1 TRANSITION
003032                          OUT     XCNTR   ; ENABLE TIROS DATA TRANSMISSION MODE
003032  323     001
```

Program 4.4.3.3(1) TIROS-N Format Transmitter Subroutine.

```
003034                                  MVI     A,60    ; TO TRANSMIT THE FIRST PHASE OF BIT 1
003034  076     060
003036                          TNO:    DCR     A       ; INSERT DELAY TO CREATE 400 BPS
003036  075
003037                                  JNZ     TNO     ; TRANSMISSION RATE
003037  302     036     006
003042                                  DLY9
003042  000
003043  177
003044                                  CMA
003044  057
003045                                  OUT     XMIT    ; TRANSMIT SECOND PHASE OF BIT 1
003045  323     000
003047                                  LXI     B,3412  ; 7 ----> B ; 12 ----> C
003047  001     012     007
003052                                  LXI     H,PADATA
003052  041     200     006
003055                                  MOV     A,M
003055  176
003056                                  MVI     E,44
003056  036     044
003060                                  DLY8
003060  000     000
003062                                  JMP     TN2
003062  303     077     006
003065                          TN1:
003065                                  DLY10           ; THIS SUBROUTINE CONTROLS
003065  177     177
                                        ;  TRANSMISSION OF THE REMAINING 14-BITS
                                        ;  OF THE BIT SYNC PATTERN, THE 9-BIT FRAME
003067                                  INX     H       ; SYNC PATTERN AND THE 24-BIT ADDRESS
003067  043
003070                                  MVI     B,10    ; CODE FROM THE 6 ROM LOCATIONS BEGINNING
003070  006     010
003072                                  MOV     A,M     ; AT ADDRESS PADATA.  AFTERWARDS THE FOUR
003072  176
003073                                  MVI     E,40    ; 8-BIT DATA BYTES ARE TRANSMITTED FROM RAM
003073  036     040
003075                                  DLY8
003075  000     000
003077                          TN2:    DLY4            ; BEGINNING AT ADDRESS TNDATA.
003077  000
003100                                  DCR     E       ; DELAY
003100  035
003101                                  JNZ     TN2
003101  302     077     006
003104                                  CMA             ; COMPLEMENT ACCUMULATOR
003104  057
003105                                  OUT     XMIT    ; TRANSMIT THE FIRST PHASE
003105  323     000
                                        ;  OF THE CURRENT LSB
003107                                  CMA             ; COMPLEMENT ACCUMULATOR
003107  057
003110                                  MVI     E,60    ; DELAY
003110  036     060
```

Program 4.4.3.3(1) (continued)

```
003112                          TN3:    DCR     E
003112  035
003113                                  JNZ     TN3
003113  302     112     006
003116                                  DLY9
003116  000
003117  177
003120                                  OUT     XMIT    ; TRANSMIT THE SECOND PHASE OF THE
003120  323     000
                                                        ; CURRENT LSB
003122                                  RRC             ; ROTATE THE DATA BYTE RIGHT ONE BIT
003122  017
003123                                  DCR     B
003123  005
003124                                  MOV     E,M     ; DELAY
003124  136
003125                                  MVI     E,45    ; IF ALL BITS OF THE CURRENT BYTE
003125  036     045
003127                                  JNZ     TN2     ; HAVE BEEN TRANSMITTED, DELAY AND
003127  302     077     006
                                                        ; THEN TRANSMIT THE NEXT BIT
003132                                  MVI     A,4     ;
003132  076     004
003134                                  DCR     C       ; IF THE FINAL DATA BYTE HAS BEEN TRANS-
003134  015
003135                                  JZ      TN5     ; MITTED, GO TO THE TRANSMITTER POWER
003135  312     155     006
                                                        ; DOWN SEQUENCE
003140                                  CMP     C       ; IF THE LAST BYTE OF ADDRESS CODE HAS
003140  271
003141                                  JZ      TN4     ; BEEN TRANSMITTED FROM ROM, PREPARE TO
003141  312     147     006
003144                                  JMP     TN1     ; TRANSMIT RAM DATA BYTES
003144  303     065     006
003147                          TN4:    LXI     H,TND-1 ; POINT DATA ADDRESS REGISTERS (HL)
003147  041     377     037
                                                        ; TO RAM DATA LOCATION MINUS 1
003152                                  JMP     TN1+2   ; AND RETURN TO TRANSMIT DATA
003152  303     067     006
003155                          TN5:    LXI     D,55    ; DELAY
003155  021     055     000
003160                                  DCR     E
003160  035
003161                                  JNZ     TN5+3
003161  302     160     006
003164                                  OUT     XCNTR   ; SHUT DOWN TRANSMITTER AND REMOVE
003164  323     001
003166                                  RET             ; POWER
003166  311
```


Program 4.4.3.3(1) (continued)

```
003167                          DELAY:
003167                                  DCX     H
003167  053
003170                                  MOV     A,L
003170  175
003171                                  ORA     H
003171  264
003172                                  JNZ     DELAY
003172  302     167     006
003175                                  DLY8
003175  000     000
003177                                  RET
003177  311


003200                          PADATA:
003200                                  DB      377     ; THE FIRST THREE BYTES OF THIS TABLE
003200  377
003201                                  DB      177     ; SPECIFY THE LAST 14 BITS OF BIT SYNC
003201  177
003202                                  DB      364     ; AND 9 BITS OF FRAME SYNC
003202  364

003203                                  DB      001     ; THE REMAINING THREE BYTES SPECIFY
003203  001
003204                                  DB      000     ; THE 24 BIT PDCP TIROS-N ADDRESS CODE
003204  000
003205                                  DB      000     ; (LSB TO MSB) - THE ADDRESS SHOWN HERE IS 1
003205  000


003206                                  END
```


Program 4.4.3.3(1) (continued)

TABLE 4.4.3.3(1)

TIROS-N DATA TRANSMISSION
SEQUENCE SPECIFICATIONS

Transmission Interval:	Variable 40, 60, or 80 seconds
Transmission Rate:	400 bits per second
Coding:	Manchester encoded with a 0 → 1 transition representing a 1

Transmission Sequence:

1.	Transmitter power-up followed by a 1 second warm-up delay	
2.	Clear carrier transmission for 160 ±2.5 milliseconds	
3.	Preamble transmission	
	a. Bit synchronization clock	(15 bits)
	b. Frame synchronization code (ØØØ1Ø1111)	(9 bits)
	c. Address code (assigned to user)	(24 bits)
4.	Data transmission - four bytes	(32 bits)
5.	Transmitter power-down	

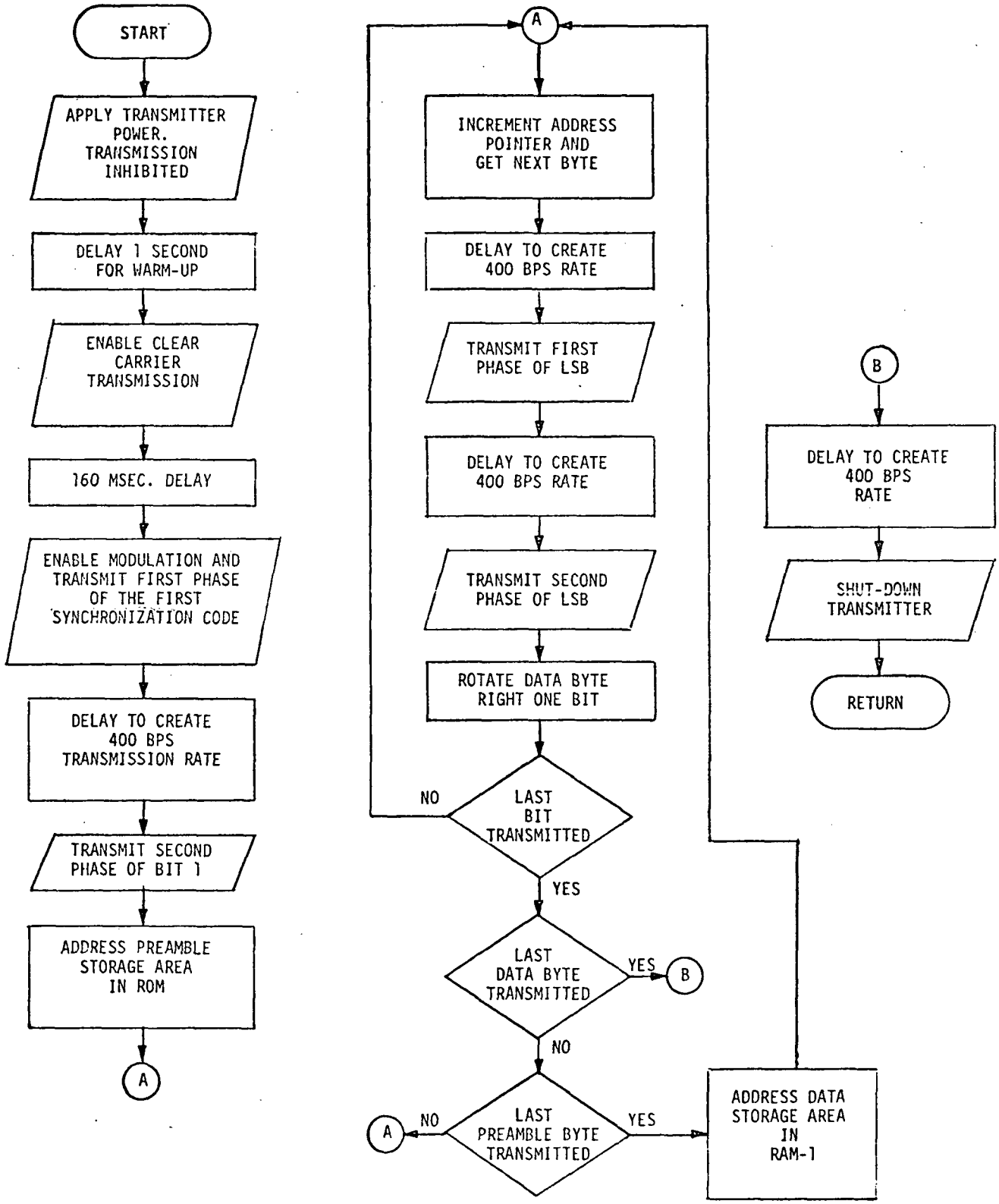


Figure 4.4.3.3(1) Flow Chart for TIROS-N Transmitter Subroutine.

defined in Section 4.4.3. The TIROS-N transmitter program utilizes eight bytes of RAM for data storage and stack operations and 124 bytes of ROM for program and preamble storage. In addition, this program also uses the standard long-delay subroutine.

4.4.3.4 Landsat Format Transmitter Subroutine - Specifications for the Landsat data transmission sequence [6] are presented in Table 4.4.3.4(1). The Landsat data transmission rate of 5000 bits per second is significantly higher than the data transmission rates used in the TWERLE, GOES, and TIROS-N data collection systems. Because of this high data transmission rate, convolutional encoding of Landsat data cannot be accomplished in real time during transmission from the UT PDCP system. Instead, a separate subroutine [Program 4.4.3.4(1)] is used to convolutionally encode the data prior to transmission. A flow chart for the PDCP convolutional encoder subroutine is illustrated in Figure 4.4.3.4(1). This subroutine is called by the executive program prior to each data transmission. The convolutional encoding process creates two code bits for each original bit of data but provides error detection and correction capabilities.

Manchester encoding, transmitter control, and actual Landsat data transmission are provided by Program 4.4.3.4(2). A flow chart for the Landsat format transmitter subroutine is shown in Figure 4.4.3.4(2). Together, Program 4.4.3.4(1) and Program 4.4.3.4(2) require 188 bytes of ROM for program storage and 31 bytes of RAM for data storage and stack operations.

4.4.4 Special Computations

Remote processing of data is a logical extension of the capabilities of the microprocessor-based PDCP. The question arises, however, as to what sort of numerical techniques are practical given the present time and memory restrictions of a PDCP system. In an effort to answer this question, the Fast Fourier Transform (FFT) was chosen for implementation. It is a sophisticated and powerful tool in numerical analysis, allowing the decomposition of time signals into their frequency components. It has been proposed, for example, that seismic events may be detected by

TABLE 4.4.3.4(1)

LANDSAT DATA TRANSMISSION SEQUENCE SPECIFICATIONS

Transmission Interval:	90 or 180 seconds
Transmission Rate:	5000 bits per second
Coding:	Convolutional encoding, rate 1/2, constraint length 5
	Manchester encoded, 1 → 0 transition representing a 1

Transmission Sequence:

1. Transmitter and platform power-up for 50 milliseconds
2. Turn on RF power simultaneous with first serial data bit
3. Data transmission

a.	Message preamble (ØØØØØØØØØØØØØØØ)	(15 zeroes)
b.	Platform address (assigned to user)	(12 bits)
c.	Sensor data 1-8 eight-bit words	(8-64 bits)
d.	Runout bits (ØØØØ)	(4 zeroes)
	Total	39-95 bits

4. Transmitter and platform power-down

NOTE: The 39-95 bits of Landsat message are before convolutional and Manchester encoding. The actual transmitted message is 4 x (39 to 95) bits.

```
                        ;              PDCP CONVOLUTIONAL ENCODER ROUTINE                ;
                        ;                                                                ;
                        ; THIS PROGRAM PERFORMS CONVOLUTIONAL ENCODING OF CONSTRAINT     ;
                        ;  LENGTH 5 ON A BLOCK OF DATA LOCATED AT THE DATA STARTING      ;
                        ;    ADDRESS.   THE ENCODING IS SUITABLE FOR ENCODING ERTS       ;
                        ;     FORMAT TRANSMITTER DATA.  FIRST BYTE OF DATA BLOCK         ;
                        ;       IS ASSUMED TO CAONTAIN THE NUMBER OF BYTES TO           ;
                        ;         BE CONVOLVED.   STACK USEAGE = 6 BYTES.               ;
                        ;                                                                ;
                        ;      CONSTANT DEFINITIONS USED IN THIS PROGRAM ARE:            ;

        000027                  S1MSK = ^B00010111      ; EXCLUSIVE OR MASK CONSTANTS
        000031                  S2MSK = ^B00011001

        004000                  DATA  = ^O4000          ; DATA BLOCK STARTS AT 2K POINT IN RAM

000000                                  ORG     1000    ; START PROGRAM AT 1/2 K POINT

001000                          CONVOLV:                ; STARTING POINT OF ROUTINE.

001000                                  LXI     D,DATA  ; POINT D AT DATA BLOCK ADDRESS
001000  021     000     010
001003                                  LDAX    D       ; #BYTES IN BLOCK ---> A
001003  032
001004                                  MOV     B,A     ; #BYTES IN BLOCK ---> B
001004  107
001005                                  XRA     A       ; CLEAR ACCUMULATOR
001005  257
001006                                  MOV     H,A     ; CLEAR H REGISTER
001006  147
001007                                  INX     D       ; LOAD NEXT THREE BYTES WITH ZERO
001007  023
001010                                  STAX    D       ;
001010  022
001011                                  INX     D       ;
001011  023
001012                                  STAX    D       ;
001012  022
001013                                  INX     D       ;
001013  023
001014                                  STAX    D       ;
001014  022
001015                                  INX     D       ; POINT DE AT LOCATION FOR HIGH ADDRESS OF PLATFORM
001015  023
001016                                  LDA     HPADR   ; LOAD ACC FROM ROM LOCATION OF HIGH ADDRESS
001016  072     142     002
001021                                  STAX    D       ; STORE HIGH ADDRESS OF PLATFORM IN <DE>
001021  022
001022                                  INX     D       ; POINT DE TO LOACTION FOR LOW ADDRESS
001022  023
001023                                  INX     D       ;
001023  023
```

Program 4.4.3.4(1) PDCP Convolutional Encoder Subroutine.

```
001024
001024  072  143  002          LDA   LPADR  ; GET HIGH PART OF PLATFORM ADDRESS
001027
001027  022                     STAX  D      ; STORE HIGH PART IN DATA BLOCK
001030
001030  021  004  010          LXI   D,DATA+4 ; POINT DE AT HIGH ADDRESS OF PLATFORM IN DATA BLOCK
001033                 CON1:
001033  023                     INX   D      ; POINT DE TO NEXT DB TO BE CONVOLVED
001034
001034  032                     LDAX  D      ; PUT NEXT DB IN ACCUMULATOR
001035
001035  157                     MOV   L,A    ; A ---> L
001036
001036  315  122  002          CALL  C4BITS ; CONVOLVE 4 BITS OF A DB INTO 8 BITS AND
                                             ; WRITE INTO LOCATION ADDRESSED BY DE.
001041
001041  023                     INX   D      ; POINT DATA BLOCK POINTER AT NEXT LOCATION
001042
001042  315  122  002          CALL  C4BITS ; NOW CONVOLVE REMAINING 4 BITS AND WRITE
                                             ; INTO NEXT LOCATION ADDRESSED BY DE.
001045
001045  005                     DCR   B      ; FINISHED CONVOLVING ENTIRE BYTE SO
                                             ; DECREMENT BYTE COUNT.
001046
001046  302  033  002          JNZ   CON1   ; NOT DONE DO ANOTHER BYTE.
001051
001051  052  000  010          LHLD  DATA   ; COMPUTE NUMBER OF BITS TO BE BIPHASED
001054
001054  051                     DAD   H      ; FROM NUMBER OF BYTES IN DATA BLOCK
001055
001055  051                     DAD   H
001056
001056  051                     DAD   H
001057
001057  051                     DAD   H      ; MULTIPLY #BYTES BY 16 BY DOUBLE ADDING HL 4 TIMES
001060
001060  021  016  000          LXI   D,14.  ; PUT 14 DECIMAL IN DE AND THEN
001063
001063  031                     DAD   D      ; ADD TO HL
001064
001064  042  000  010          SHLD  DATA   ; NUMBER OF BITS TO BE ENCODED INTO FIRST BYTE OF DATA BLOCK
001067
001067  311                     RET          ; ALL DONE, RETURN TO EXECUTIVE
```


Program 4.4.3.4(1) (continued)

```
                                        ;       SUBROUTINES     ;

001070                          NEXTB:                  ; ROUTINE TO CONVOLVE ONE BIT INTO
                                                        ; TWO BITS AND SAVE RESULT IN C

001070                                  DAD     H       ; SUB-SUBROUTINE TO ENCODE A BIT-FIRST SHIFT
001070  051
                                                        ; DB LEFT ONE BIT INTO H REGISTER

001071                          NDB1:   MOV     A,H     ; PUT SHIFTED BITS IN ACCUMULATOR
001071  174
001072                                  ANI     S2MSK   ; PICK OFF BITS TO EXCLUSIVE OR
001072  346     031
001074                                  MVI     A,0     ; CLEAR A BUT DON'T AFFECT FLAGS
001074  076     000
001076                                  JPE     NDB2    ; IF EVEN NUMBERS OF ZEROES FOR S2
001076  352     102     002
                                                        ; CALCULATION, LEAVE A=0

001101                                  INR     A       ; ELSE PUT A ONE IN ACCUMULATOR
001101  074
001102                          NDB2:   ORA     C       ; "OR" S2 WITH PREVIOUSLY CONVOLVED BITS
001102  261
001103                                  MOV     C,A     ; AND SAVE IN C
001103  117
001104                                  MOV     A,H     ; GET SHIFTED BITS FROM H AGAIN
001104  174
001105                                  ANI     S1MSK   ; MASK S1 BITS
001105  346     027
001107                                  MVI     A,2     ; PUT A ONE IN SECOND BIT POSITION OF ACCUMULATOR
001107  076     002
001111                                  JPO     NDB3    ; LEAVE THE BIT=1 IF ODD NUMBER OF ZEROES
001111  342     115     002
001114                                  XRA     A       ; ELSE CLEAR ACCUMULATOR
001114  257
001115                          NDB3:   ORA     C       ; "OR" S1 WITH PREVIOUSLY CONVOLVED BITS
001115  261
001116                                  RRC             ; AND SHIFT BITS RIGHT TWO PLACES
001116  017
001117                                  RRC             ;
001117  017
001120                                  MOV     C,A     ; SAVE CONVOLVED BITS IN C
001120  117
001121                                  RET             ; RETURN TO CALLING POINT
001121  311

001122                          C4BITS:                 ; ROUTINE TO CONVOLVE 4-BITS INTO 8-BITS
                                                        ; 8-BIT RESULT IS STORED IN LOCATION
                                                        ; PRESENTLY ADDRESSED BY DE PLUS ONE.
001122                                  MVI     C,0     ; CLEAR C FIRST
001122  016     000
001124                                  CALL    NEXTB   ; CONVOLVE ONE BIT
001124  315     070     002
001127                                  CALL    NEXTB   ; CONVOLVE ANOTHER BIT
001127  315     070     002
```

Program 4.4.3.4(1) (continued)

```
001132                                  CALL    NEXTB   ; CONVOLVE ANOTHER BIT
001132  315     070     002
001135                                  CALL    NEXTB   ; CONVOLVE FINAL BIT FOR THIS ROUTINE
001135  315     070     002
001140                                  STAX    D       ; SAVE CONVOLVED BITS IN PRESENT DE LOCATION
001140  022
001141                                  RET             ;
001141  311

001142                          HPADR:  DB      10      ; HIGH FOR BITS OF PLATFORM ADDRESS WOULD BE
001142  010                                             ; STORED IN ROM HERE

001143                          LPADR:  DB      00      ; LOW 8 BITS OF PLATFORM ADDRESS STORED HERE
001143  000
001144                                  END
```

Program 4.4.3.4(1) (continued)

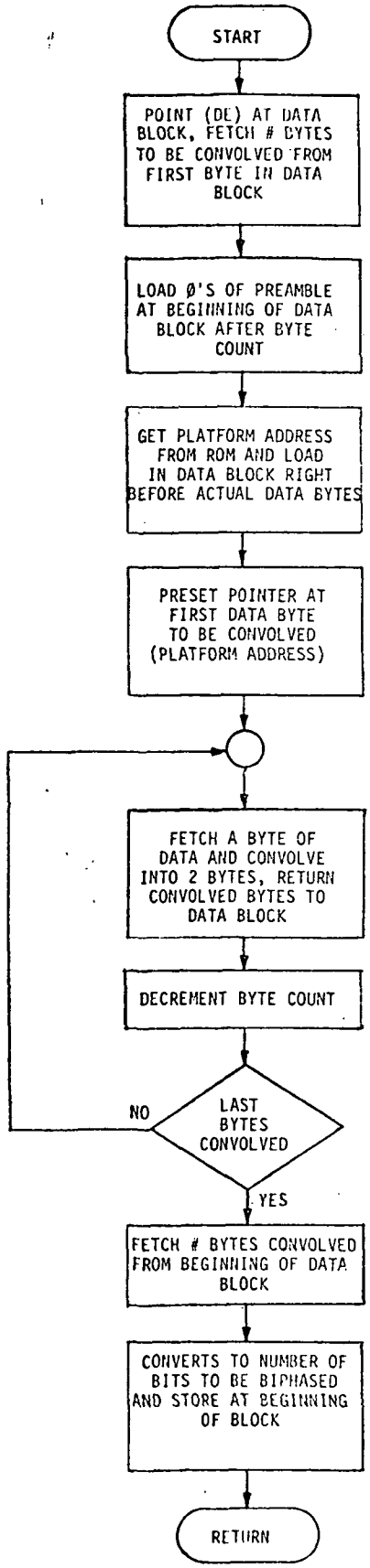


Figure 4.4.3.4(1) Flow Chart for the PDCP Convolutional Encoder Subroutine.

```
                        ;            LANDSAT FORMATTED TRANSMITTER ROUTINE          ;
                        ;                                                           ;
                        ; THIS PROGRAM CONTROLS THE TRANSMISSION OF UP TO 190 BITS OF ;
                        ;  LANDSAT FORMATTED DATA MESSAGE USING AN 8080A MICROPROCES- ;
                        ;   SOR WITH A 1.6666 MICROSECOND STATE TIME.   ALL TIMING    ;
                        ;    FUNCTIONS ARE ACCOMPLISHED IN SOFTWARE.  "LDLY" IS A    ;
                        ;    MACRO INSTRUCTION WHICH CAUSES A SOFTWARE DELAY OF      ;
                        ;     50 MILLISECONDS BETWEEN OUTPUTS.                      ;
                        ;                                                           ;
                        ;    DATA CONSTANTS USED ARE DEFINED BELOW:                 ;

004000                          DATA=4000               ; DATA BLOCK AT 2K POINT

000062                          PWRON= ^B00110010       ; CONTROL WORD CONSTANTS
000026                          RFON=  ^B00010110
000040                          PWROFF=^B00100000

000000                          XMIT=0                  ; OUTPUT PORT ASSIGNMENTS
000001                          XCNTR=1

000000                          ORG     2000    ; START PROGRAM AT 1 K POINT

002000                  LXMIT:                  ; START OF LANDSAT TRANSMITTER ROUTINE.

002000                          LXI     H,DATA  ; POINT MEMORY AT DATA BLOCK, 2K POINT
002000  041    000     010
002003                          MOV     D,M     ; #BITS TO BE BIPHASED ---> D
002003  126
                                                ; D = NUMBER OF CONVOLVED BYTES * 16 +14
002004                          MVI     C,10    ; 8 DECIMAL ---> C
002004  016    010
002006                          MVI     A,PWRON ; SET UP CONTROL WORD TO TURN ON ALL PWR EXCEPT RF
002006  076    062
002010                  LXM1:   OUT     XCNTR   ; POWER ON!
002010  323    001
                                                ; 50 MILLISECONDS LATER TURN ON RF
002012                          INX     H       ; POINT TO FIRST DATA BYTE
002012  043
002013                          MOV     A,M     ; FIRST DB ---> ACCUMULATOR
002013  176
002014                          OUT     XMIT    ; PUT FIRST BIT OUT SO DATA WILL
002014  323    000
                                                ; BE TRUE WHEN RF COMES ON
002016                          MVI     E,3     ; SET E TO BIPHASE ONLY 3 BITS OF FIRST BYTE
002016  036    003
```

Program 4.4.3.4(2) Landsat Format Transmitter Subroutine.

```
002020                                  LDLY            ; DELAY 50 MILLISECONDS FROM POWER ON TO RF ON
002020  345
002021  041     336     004
002024  053
002025  175
002026  264
002027  302     024     004
002032  341
002033  177     177
002035  000
002036  177
002037                                  MVI     A,RFON  ; SET UP CONTROL WORD TO OPERATE
002037  076     026
                                        ;               IN LANDSAT MODE WITH RF ON, DATA ENABLED AND
                                        ;               INTEGRATOR INITIAL CONDITION RELEASED
002041                          LXM2:   OUT     XCNTR   ; RF ON AND START DATA FLOW!
002041  323     001
002043                                  MOV     A,M     ; RESTORE FIRST DATA BIT OF FIRST BYTE
002043  176
002044                                  DLY29           ; TIME THE BIT FOR 60 MACHINE STATES
002044  305     301
002046  000     000
002050                                  JMP     AG1     ; FROM LXM2: TO AG1:
002050  303     066     004
002053                          NEXT:   INX     H       ; POINT MEMORY TO NEXT DB
002053  043
002054                                  MOV     A,M     ; PUT NEXT DB --> ACCUMULATOR
002054  176
002055                                  MOV     E,C     ; SET BIT COUNT IN E TO 8 DECIMAL
002055  131
002056                                  DCR     D       ; DECREMENT BIT COUNT
002056  025
002057                                  DLY5            ; DELAY 50 BIT TIME IS 60 MACHINE STATES
002057  177

002060                          AGAIN:  OUT     XMIT    ; OUTPUT FIRST PHASE OF BIT
002060  323     000
002062                                  DLY46           ; DELAY 50 BIT TIME IS 60 MACHINE STATES
002062  177     177
002064  343     343
```


Program 4.4.3.4(2) (continued)

```
002066                          AG1:    CMA             ; COMPLEMENT BIT TO SEND SECOND PHASE OF BIT
002066  057
002067                                  OUT     XMIT    ; OUTPUT SECOND PHASE OF BIT
002067  323     000
002071                                  CMA             ; RESTORE BIT TO TRUE FORM
002071  057
002072                                  RRC             ; SHIFT BIT RIGHT ONE BIT POSITION
002072  017
002073                                  DCR     E       ; DECREMENT BIT COUNT
002073  035
002074                                  JZ      NEXT    ; AFTER 8 BITS GO GET NEXT DB
002074  312     053     004
002077                                  DCR     D       ; DECREMENT TOTAL BIT COUNT (CHECK FOR NEXT
002077  025
                                                        ; TO LAST BIT OUTPUTTED)
002100                                  DLY12           ; MAKE LOOP = 60 MACHINE STATES
002100  000     000
002102  000
002103                                  JNZ     AGAIN   ; OUTPUT ANOTHER BIT IF NOT THROUGH
002103  302     060     004
002106                                  OUT     XMIT    ; OUTPUT LAST BIT, FIRST PHASE
002106  323     000
002110                                  DLY46           ; DELAY FOR BIT TIME OF 60 MACHINE STATES
002110  177     177
002112  343     343
002114                                  CMA             ; COMPLEMENT BIT TO SEND SECOND PHASE
002114  057
002115                                  OUT     XMIT    ; OUTPUT SECOND PHASE OF LAST BIT
002115  323     000
002117                          SHUTDW: MVI     A,PWROFF ; LOAD ACCUMULATOR WITH SHUTDOWN CONTROL WORD.
002117  076     040
002121                                  DLY43           ; DELAY SO BIT TIME IS 60 MACHINE STATES
002121  177     177
002123  305     301
002125  000     000
002127  000
002130                                  OUT     XCNTR   ; SHUTDOWN PLATFORM POWER
002130  323     001
002132                                  RET             ; ALL DONE, RETURN TO EXECUTIVE.
002132  311

002133                                  END
```


Program 4.4.3.4(2) (continued)

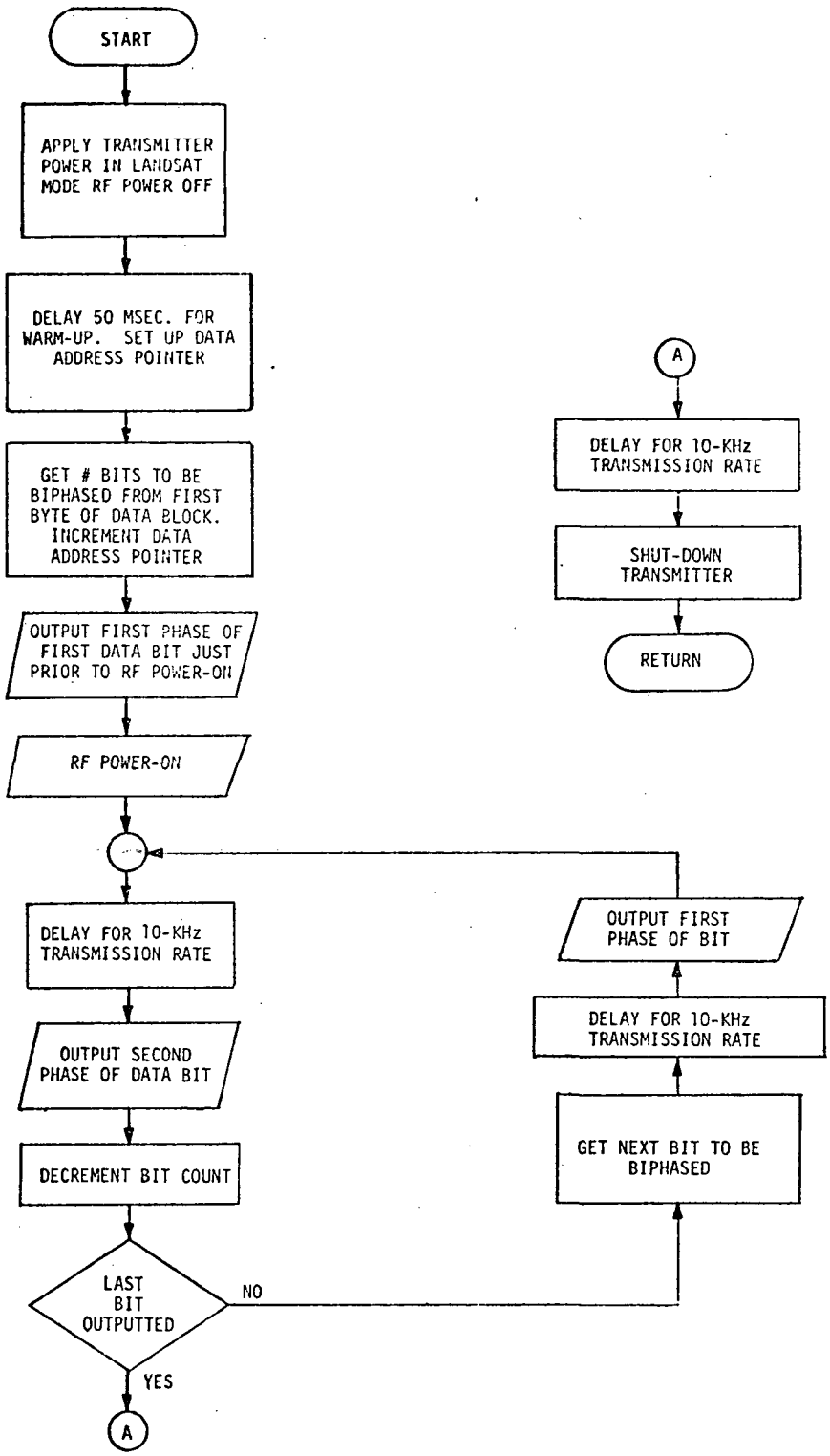


Figure 4.4.3.4(2) Flow Chart for the Landsat Transmitter Subroutine.

an analysis of their frequency components, and with this in mind a classifier based on the Bayes decision rule was investigated for its possible practical application to seismic event detection. Figure 4.4.4(1) shows how the FFT and Bayes classifier would work together in a detection scheme.

4.4.4.1 The FFT Program - The implementation of the fast Fourier transform (FFT) program was carefully studied. Because of limitations in execution time and memory size imposed by the PDCP, the commonly used forms of the algorithm were unacceptable. Most involve considerable manipulation of complex numbers, in particular, complex multiplication. Also, most existing FFT programs require the calculation of weighting factors, which are themselves complex and trigonometric in nature.

A method which avoids these time consuming problems was reported in a short paper by Dr. C. M. Rader and N. M. Brenner [9]. It outlines an approach whereby only pure imaginary weighting factors are required, thus decreasing the total number of multiplications necessary. In addition, the weighting factors fall in a convenient range of values, from j0.5 to j5 (for 64-point transforms or less).

The derivation of the Rader-Brenner form is outlined in Appendix B and need not be repeated here. However, it is helpful to see the flow of the process. Figure 4.4.4(2) outlines the sequence of events graphically for each data byte pair (complex data value) of a 16-point transform. First is the summation stage where c_n is calculated for the two-point, four-point, and eight-point cases. Recall from the derivation that $c_n = a_{2N+1} + a_{2N-1} + Q$, where n = 0, 1, ..., N/2-1. Next, the data values are permuted so that the final output data will be in numerical and not bit-reverse order [10]. That is, cell one will contain the first Fourier coefficient, cell two will contain the second, and so on. Finally, the two-, four-, eight- and 16-point transforms are calculated.

As can be seen from the 8080A FFT program source listing, (Program C(1) in Appendix C) a FORTRAN program is used as the logical model for the assembly language program. There are several advantages to this approach. In the first place, FORTRAN offers a clear way to logically

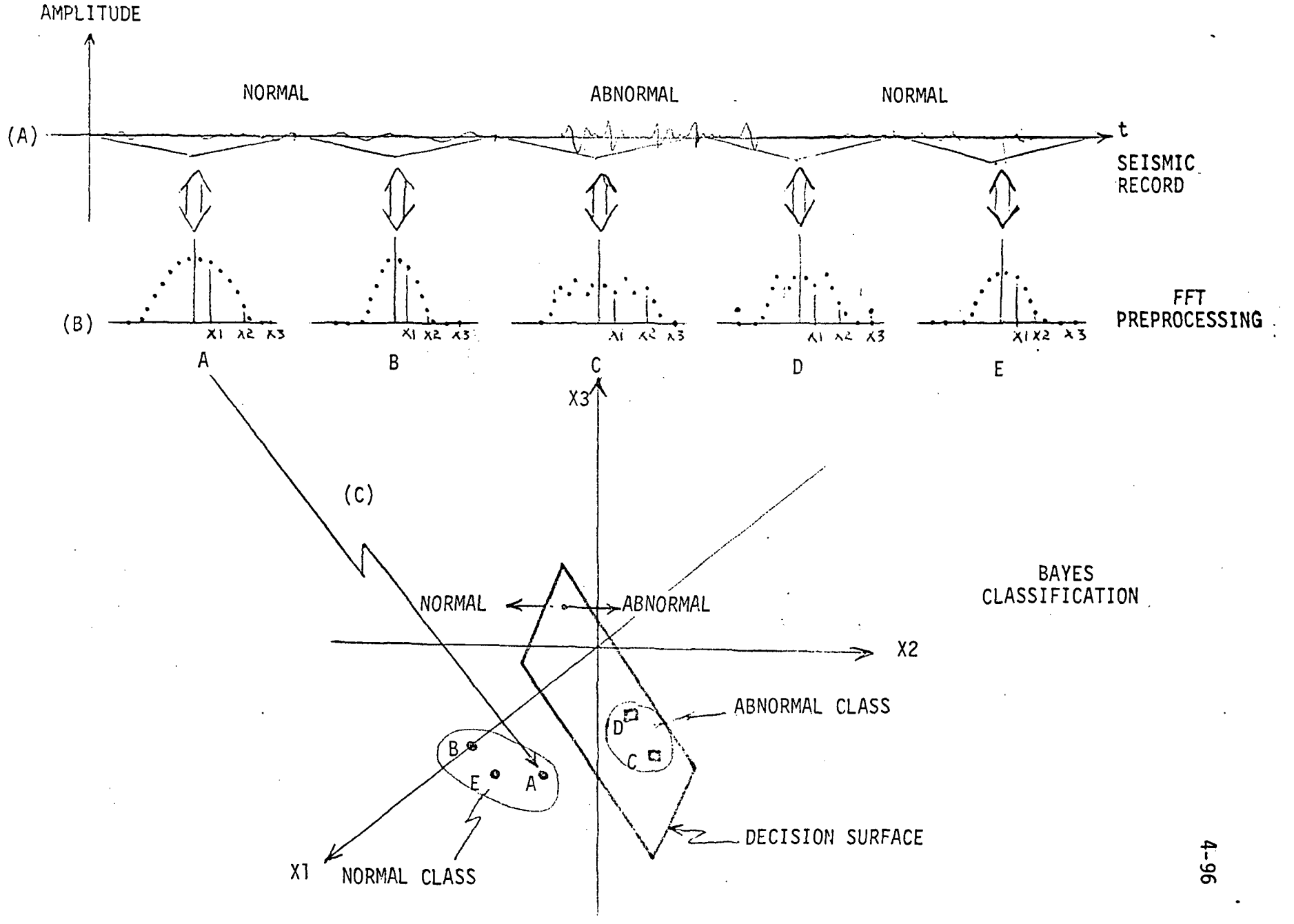


Figure 4.4.4(1) Seismic Record Preprocessing and Classification.

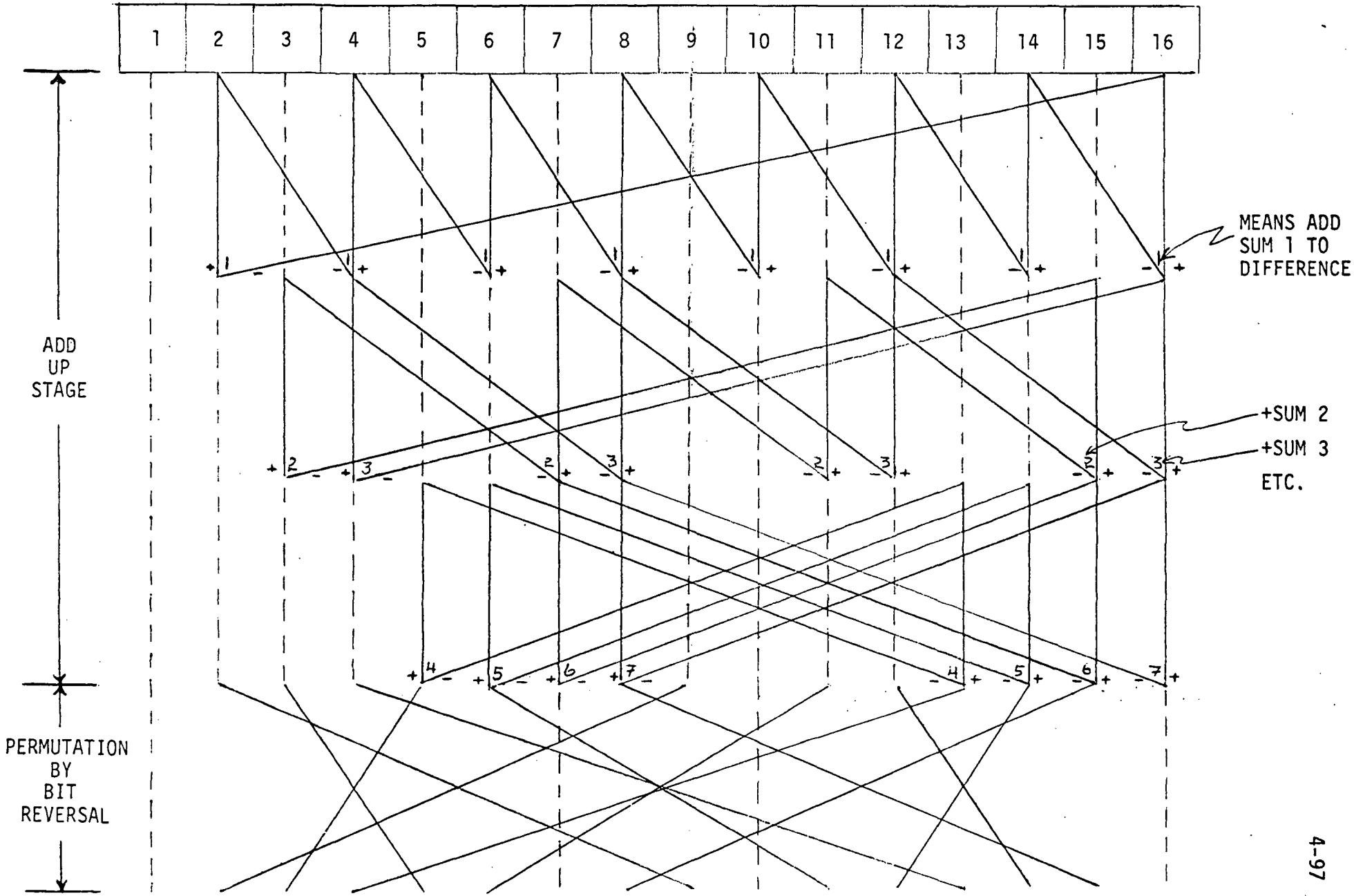


Figure 4.4.4(2) FFT Data Flow Chart

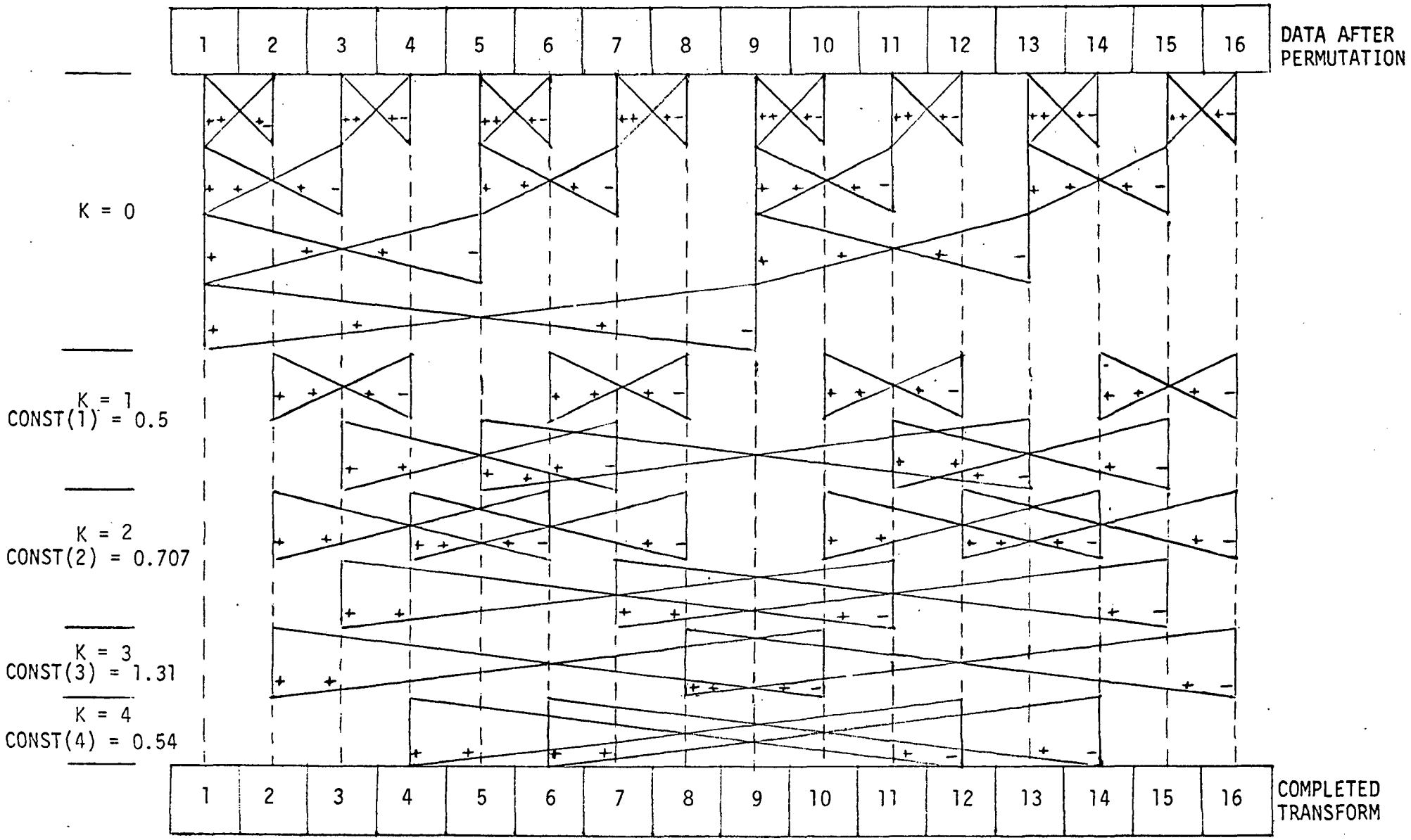


Figure 4.4.4(2) (continued)

*Multiply term by (0,1)*CONST(K).

break down the operations of the program. Secondly, a program written in FORTRAN (with certain hard-to-implement instructions like DO omitted) can be reduced to minimum complexity and debugged on a logical level before assembly language programming begins. Thus, one is assured of the integrity of the logic before proceeding to assembly language programming. The 8080A fast Fourier transform subroutine (FFT80) requires 1102 bytes of ROM.

After the FORTRAN framework is decided upon, compilation to assembly language can begin. However, assembly by a FORTRAN compiler would be quite unsatisfactory. Very great increases in efficiency and speed can be achieved by taking advantage of the architecture of the 8080A CPU. For example, instead of storing commonly used program variables in memory, which would require time consuming memory read/write operations, these values can be retained and operated upon using internal registers. It is, of course, occasionally necessary to free registers for other uses, which is achieved by pushing and popping the variables on the stack. The total time required for these operations is small compared to the time required for repeated memory accesses.

It would have been preferable to use a word length of 16 bits in this program as it allows greater resolution of incoming signals and removes the fear of overflow errors. This would mean complex data values would be 32 bits long. Unfortunately, the 8080A's double word (16 bit) instruction set is quite limited and very time consuming, while separately manipulating each byte of a double word is even more inefficient. The only reasonable approach then is to use 8-bit values, thus trading resolution for speed. In a few intermediate steps of the program it is necessary to expand to double words, such as in manipulation of the variable SUM in the add up stage of the program. SUM is a mean value which first contains a sum of values and then is divided by the number of values summed. Before division, SUM can easily grow to a value in excess of eight bits in length, so the word length is increased. After division, of course, eight bits is sufficient since an average cannot be larger than the largest of the averaged values.

The 8080A does not provide hardware multiplication, and conventional software multiplication is quite time consuming especially considering the non-integer factors involved. Thus, an alternate multiplication technique has been developed. From the development in Appendix B, it is seen that the weighting factors are pure imaginary and have the values:

$$w_k = 0.5 \text{ cosecant}(2\pi k/N)j, \qquad k \neq 0,N/2 .$$

In Program C(1), 2k is defined M1 and N is defined M2. That is, w_k = 0.5 cosecant(πM1/M2)j. The sequence M1/M2 is:

$$M1/M2 = m/2^n ,$$

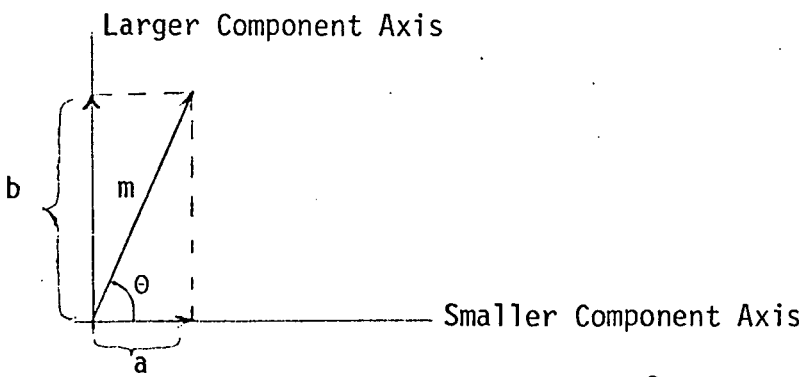
where m = 1, 3, 5, ..., 2^{n-1}-1 for each n = 1, 2, 3, ..., ($\log_2 N$)-1 (where N = number of points of the transform). For a 64-point transform, M1/M2 is the sequence 1/2, 1/4, 1/8, 3/8, 1/16, 3/16, 5/16, 7/16, 1/32, 3/32, 5/32, 7/32, 9/32, 11/32, 13/32, 15/32. A 16-point transform would require only the sequence M1/M2 = 1/2, 1/4, 1/8, 3/8. Table 4.4.4(1) delineates the ratios and consequent values of w_k.

The shifts listed in the table are used by the special multiplication subroutine MULCON (TEMP,K). MULCON multiplies by shifting the value in TEMP right or left a number of times and adding or subtracting the shifted value with the original value of TEMP. The number of shifts and the decision as to whether the resulting shifted value is to be added to or subtracted from TEMP is stored in a look-up table (called TABLE in the program). K is the index for TABLE. For example, suppose K = 10_8 when MULCON is called. M1/M2 = 3/8 and w_k = 0.54. The value in TEMP is to be multiplied by 0.54, thus TABLE (10_8) = -1,+5. TEMP is duplicated in TMPROD and shifting proceeds on TMPROD. The 1 value in TABLE indicates that TEMP is to be shifted right one time, yielding 0.5 TEMP, and the minus sign means that the shifted value is to be subtracted from TMPROD. Recalling that TMPROD = TEMP originally, we now have TMPROD = TMPROD - TEMP/2^1 = 0.5 TEMP. Next, TEMP is shifted right five times and the shifted value is added to TMPROD, which equals 0.5 TEMP from the last operation. Five

shifts yield 0.031 TEMP and TMPROD = TMPROD + TEMP/2^5 = 0.5 TEMP + 0.031 TEMP = 0.53 TEMP. From Table 4.4.4(1) it can be seen that the exact answer is 0.54; the error is less than 0.01 TEMP. This is usually less than round-off error when working with integer arithmetic.

Upon completion of the FFT portion of the program, the array DATA, which initially held the time domain data, now contains the Fourier series coefficients. These transform values, however, are complex and for the purposes of Bayes classification, only magnitudes are of interest. The normal procedure for finding the magnitude of some value (a+jb) is to take the square root of the sum of the squares of the coefficients, that is, $\sqrt{a^2 + b^2}$. To speed up this computation for a microprocessor application, a method is used which does not require the squaring and square-root routines; only the division routine is required. This method is described in the following paragraph.

A complex number can be visualized as a vector of magnitude m having two orthogonal components. Since m is always positive, the signs of these components are irrelevant; only the absolute values need be considered. The absolute value of the smaller component is defined here as a and the larger as b, so m is graphed.



It is clear that for these conditions, $45^0 \leq \theta \leq 90^0$. What is needed is some function, C(a,b), such that a*C = m, that is, some function that relates the smaller component to the magnitude. It is apparent that a*secant(θ) = m, but this is a function of θ. The next step is to express θ as a function of a and b. Of course, θ = arctan(b/a), so the above expression becomes:

$$a*\text{secant}[\arctan(b/a)] = m, \qquad 1 \leq b/a < \infty .$$

TABLE 4.4.4(1)

MULTIPLICATION (MULCON) CONSTANTS

$K_{10}(K_8)$	M1/M2	w_K^*	Right Shift	Left Shift
1 (2)	1/2	0.500j	-1,0	
2 (4)	1/4	0.707j	-2,-5	
3 (6)	1/8	1.307j	+2,+4	
4 (10)	3/8	0.541j	-1,+5	
5 (12)	1/16	2.563j	+1,+4	Shift left but do not add to original
6 (14)	3/16	0.900j	-4,-5	
7 (16)	5/16	0.601j	-1,+3	
8 (20)	7/16	0.510j	-1,0	
9 (22)	1/32	5.101j	+3,0	+2
10 (24)	3/32	1.722j	+1,+2	
11 (26)	5/32	1.061j	+4,0	
12 (30)	7/32	0.788j	-2,+5	
13 (32)	9/32	0.647j	-1,+3	
14 (34)	11/32	0.567j	-1,+4	
15 (36)	13/32	0.523j	-1,+5	
16 (40)	15/32	0.502j	-1,0	

$^*w_K$ = 0.5 cosecant(πM1/M2)j

Thus, C = secant[arctan(b/a)]. Now the calculation of C for each b and a would be as difficult as the RMS method, but this calculation is not necessary. Instead, a "window" technique is used, whereby a value C is associated with a certain range of values of b/a. In the actual program, the value 8b/a is calculated using a standard division routine as this preserves more significant digits in the quotient.

Because it is desired to perform the effective "multiplication" of C*a by successive additions of a, integral values of C are desired. Further, since the value 0.5a can be easily found and stored, then C can take the values 1.5, 2, 2.5, 3, ..., 9, 9.5.

What must now be found is the range of values of b/a for which a particular value of C will be chosen. It seems reasonable, where the possible values of C are 2, 2.5, 3, 3.5, etc., that those values of b/a which yield a $2.25 \leq C < 2.75$ should be assigned the value C = 2.5, or where $2.75 \leq C < 3.25$, then C = 3. Using this method, the absolute error in the resulting magnitude is less than 0.25 times the smallest component. The magnitude, therefore, is an approximation; however, this approximation is adequate for most purposes.

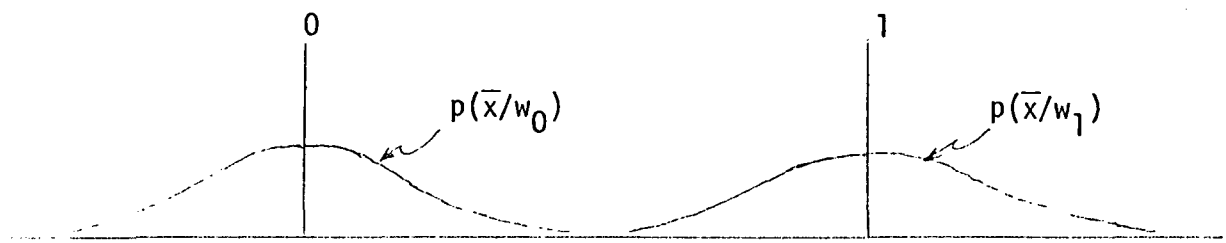
The FFT program composes what can be called the preprocessor part of the classification scheme. It receives the incoming data and finds in it certain parameters or features of interest, in this case frequency components. The next step is to extract those features which can be used to assign the data to a particular class. For FFT80, feature extraction merely means taking certain pre-determined complex frequency components, finding their magnitudes, and storing these magnitudes in a storage array, called MTUDE. The contents of the MTUDE array become the input data to the pattern recognizer stage of the program.

4.4.4.2 The Bayes Classifier - The classification scheme investigated uses the Bayes decision rule. In order to understand how this process works, it is helpful to visualize a space in which every pattern (input value) is a point in space. Thus, the space has as many dimensions as the pattern. For example, suppose the three low frequency components

X1, X2, and X3 are being used to classify a signal. Then for each frame pattern, $\bar{x}$ has three components as seen in Figure 4.4.4(1). Each point, $X(F_2,F_3,F_4)$, represents a pattern in three-space. If, after a sufficient number of trials (points plotted), the points are observed to cluster in groups, the Bayes decision rule can be applied. The clustered groups are called classes and some decision must be made as to what constitutes a class boundary. Once the boundaries are defined, each succeeding pattern will be classified according to which side of a boundary it falls upon. The Bayes method does not assure that misclassifications will not be made, but it does assure a minimum average loss in classification; that is, it assures statistical optimization [11]. The average loss in assigning a pattern $\bar{x}$ to a class j given m classes is

$$r_j(\underline{x}) = \sum_{i=1}^{m} L_{ij} p(\bar{x}/w_i)\ p(w_i)$$

where $p(\bar{x}/w_i)$ is the probability density function of w_i, $p(w_i)$ is the probability of the occurrence of the class w_i and L_{ij} is a "loss matrix" which assigns a loss of zero for correct classifications and a loss of one for misclassifications. A pattern is assigned to the class offering the smallest average loss. Consider a simple example. Suppose we are receiving a signal that has been corrupted by Gaussian noise. The signal is made up of 1's and 0's, so the actual received signal has a probability density



while $p(w_0)$ is the probability of a 0 being sent and $p(w_1)$ is the probability of a 1 being sent. A reliable classifier can be implemented only if the two density curves do not significantly overlap.

In separating seismic signals into normal (background noise) and abnormal (seismic activity) classes, based on spectral composition, several things must be known. The values of p(normal) and p(abnormal) are not critical and can be arbitrarily set to p(normal) = 0.9999 and p(abnormal) = 0.0001. However, calculating values for the probability densities of normal and abnormal classes is more difficult. To an extent these are functions of the location of the seismic sensor. A PDCP located near a highway, for example, would have a noise spectrum and probability density different from one located in a more remote area. The factors affecting the probability density of seismic spectral components are not well understood and there is a dearth of information on the subject [12]. But perhaps the greatest problem encountered in the use of the Bayes classifier on spectral components of signals is that the signals of both classes are largely impulsive in nature. Impulses, when transformed, exhibit all frequencies equally, and, seismic and noise signals, while not pure impulses, are similarly sharp, fast-changing time functions. The problem then is that any given combination of frequencies that can be extracted from PDCP processing will be similar for both noise and seismic signals. That is, their probability density curves overlap to an extent that they cannot be reliably separated.

It is possible, however, to use the Bayes classifier to separate patterns with high frequency (impulsive-type) components from those with only low frequency components. This would enable noise and seismic signals to be differentiated from calm background activity; but if this were all that was required, a simple high pass filter would be the obvious choice.

While the use of Bayes classification of spectral data cannot be removed from consideration, it is suggested that to be of practical value very high resolution (multiple point) transforms are required. In addition, PDCP's must be trained individually with data taken from the sites on which they will be placed.

The implementation of a fast Fourier transform subroutine on an 8080A microcomputer requires the addition of 1.1K of memory. It will execute 16 point transforms in 39 ms, 32 point transforms in 96 ms, and 64 point

transforms in 228 ms at a CPU clock rate of 2 mHz. This demonstrates the feasibility of performing sophisticated and complex mathematical functions remotely on PDCP. With the advent of new microprocessors with increased speed, indirect addressing, and hardware multiply/divide, such as the Texas Instruments TMS 9900, multipoint FFT subroutines will become practical.

REFERENCES

1. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, 1973.

2. "Intel 8080 Microcomputer Systems User's Manual," Intel Corporation, July 1975.

3. Lathi, B. P., Communication Systems, John Wiley & Sons, Inc., New York, New York, 1965, pp. 89-91.

4. Hogg, G., "A Data Compression Primer," X-Document X-521-65-320, NASA - Goddard Space Flight Center, Greenbelt, Maryland, August 1965.

5. Kostopulos, G. K., Digital Engineering, John Wiley & Sons, Inc., New York, New York, 1975, pp. 309-311.

6. Schoep, V. R., "ERTS/GOES Convertible Data Collection Platform," Ball Brothers Research Corporation, Final Report for U. S. Geological Survey EROS Program Office Contract 14-08-0001-1390, 1974.

7. Lichfield, E. W. and Carlson, N. E., "Tropical Wind, Energy Conversion and Reference Level Experiment (TWERLE)," Engineering Design Report No. 2, National Center for Atmospheric Research, November 1974.

8. Coates, J. L., "The Nimbus-F Random Access Measurement System (RAMS)," IEEE Transactions on Geoscience Electronics, Vol. GE-13, No. 1, January 1975, pp. 18-27.

9. Rader, C. M. and N. M. Brenner, "A New Principle for Fast Fourier Transformation," Unpublished notes, 1972.

10. Brigham, Oran E., The Fast Fourier Transform, Englewood Cliffs, New Jersey: Prentice-Hall, 1974, p. 175.

11. Tou, J. T. and R. C. Gonzalez, Pattern Recognition Principles, Reading, Massachusetts: Addison-Wesley Publishing Co., 1974, p. 113.

12. Bath, Markus, Introduction to Seismology, New York: Halsted Press, 1973, pp. 249-251.

5. PROGRAMMABLE DATA COLLECTION PLATFORM DEVELOPMENT SYSTEM

The University of Tennessee programmable data collection platform development system includes an Intel 8080 based special-purpose microcomputer, a video display terminal, a cassette bulk storage device, and supportive system software.

5.1 UT PDCP DEVELOPMENT SYSTEM HARDWARE

A major objective of the PDCP study is to conceive and describe a programmable data collection platform. To aid in achieving this goal, the UT PDCP development system using the Intel 8080 microprocessor has been built. The UT PDCP is designed to serve as a developmental tool which will aid in determining the feasibility of using a microprocessor to perform all current DCP tasks as well as additional tasks which are not possible with contemporary DCP designs. The UT PDCP includes the capability of simulating the transmitted data messages for typical Landsat, GOES, TIROS-N and TWERLE data collection platforms. In addition, the UT PDCP is interfaced to a video display terminal and keyboard permitting user intervention with the various PDCP demonstration programs. The UT PDCP is not intended to reflect a choice of microprocessors for an actual PDCP; this choice will depend on many factors which are described in Section 6.

The basic hardware system consists of eight 11.43cm x 16.51cm (4.5" x 6.5") printed circuit cards manufactured by Pro-Log Corporation. There are three 4K RAM cards, two 2K ROM cards, one 32-line input card, one 32-line output card, and the CPU card. The eight Pro-Log MPS components cards supply 32 parallel input and output lines for interfacing simulated sensors, the system memory (12K RAM and up to 4K ROM), and the microprocessor chip and support logic. To complete the UT PDCP, four printed circuit cards have been added to the basic Pro-Log system. The additional cards include a dual serial I/O card, an analog interface card, a cassette analog interface and baud-rate timing card, and a utility card containing hardware system start-up, a baud-rate clock and current loop

interface for the video terminal, and single step control logic. All printed circuit cards plug into a small 12.7cm x 25.4cm (5" x 10") rack with a wire-wrapped back plane. The rack is mounted inside an attractive gray cabinet. Power is provided by a small external power supply. The front panel switches are minimal as control of the UT PDCP is intended to be from keyboard commands issued from the video terminal. Jacks for interconnection of the video terminal and cassette recorder are on the rear panel. Simulation of several DCP functions is provided by LED's mounted on the front panel. All connections between the cassette recorder, video display and the UT PDCP are via plug-in cables for ease of interconnection of the three system components. The only additional hardware needed to completely demonstrate the capabilities of the UT PDCP system are a laboratory dual-sweep oscilloscope and a frequency counter with period measurement capability. A description of the operating procedure for these instruments is included in the System Operation Manual. The following sections describe in detail the individual components of the UT PDCP system. A block diagram of the major components of the UT PDCP system hardware is depicted in Figure 5.1(1). Figure 5.1(2) presents a more detailed view of the PDCP microcomputer system components.

5.1.1 Basic Pro-Log System

An 8080 based microprocessor system was selected for the UT PDCP. There are several reasons for this choice. The 8080 microprocessor has an eight bit word size which should be optimal for the PDCP application. Extensive software support packages including resident text editors, assemblers, simulators, BASIC, FOCAL, and FORTRAN are available for the 8080. In addition, cross-assemblers, a cross-simulator, and a FORTRAN cross-compiler are also available. A few recent microprocessors have extensive software support available now; however, at the beginning of this study only the 8080A had extensive software support. Due to the short period allotted for this study, extensive software support was required to enable development of a large number of PDCP programs.

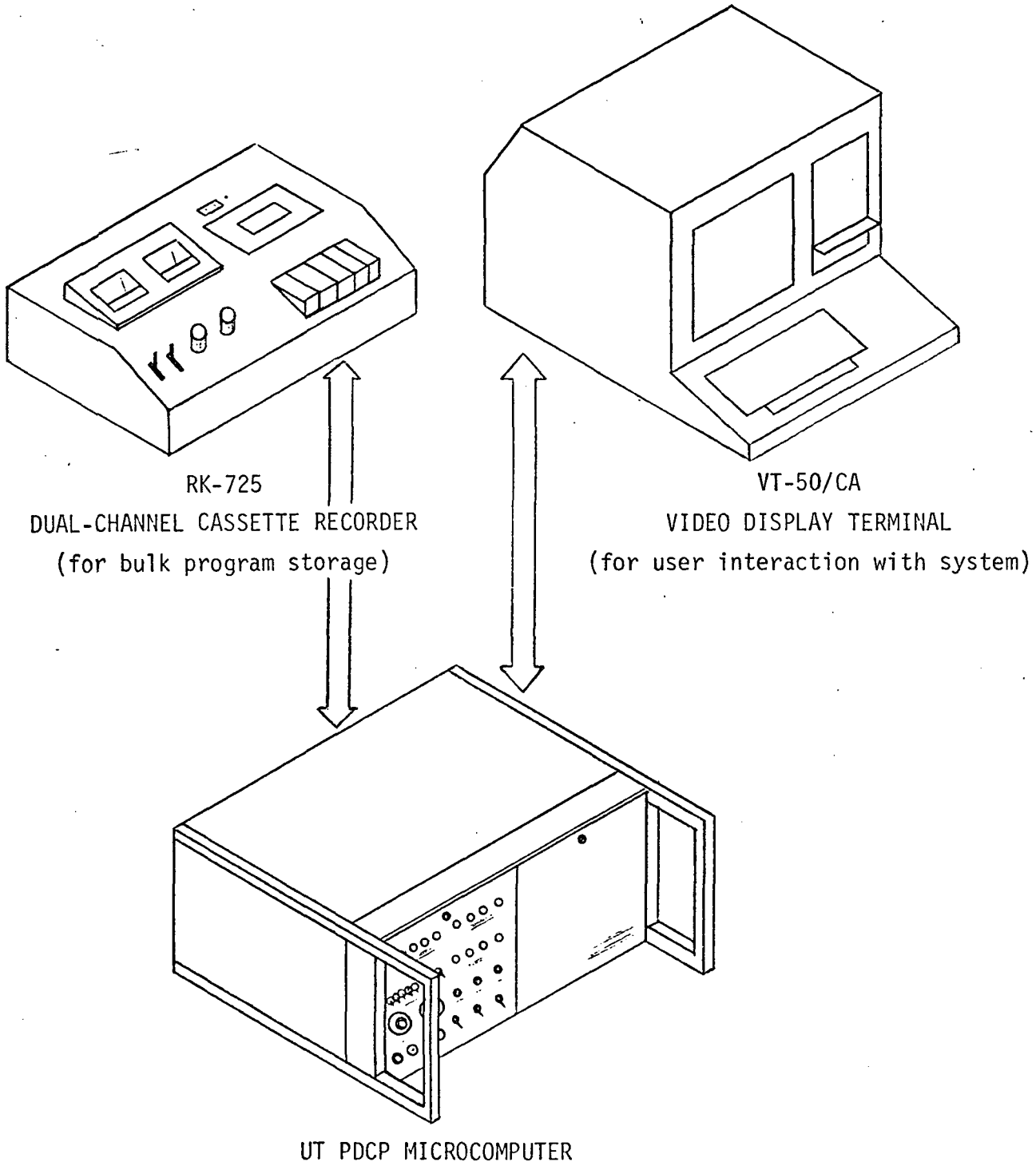


Figure 5.1(1) Major Components of the UT PDCP DEVELOPMENT SYSTEM.

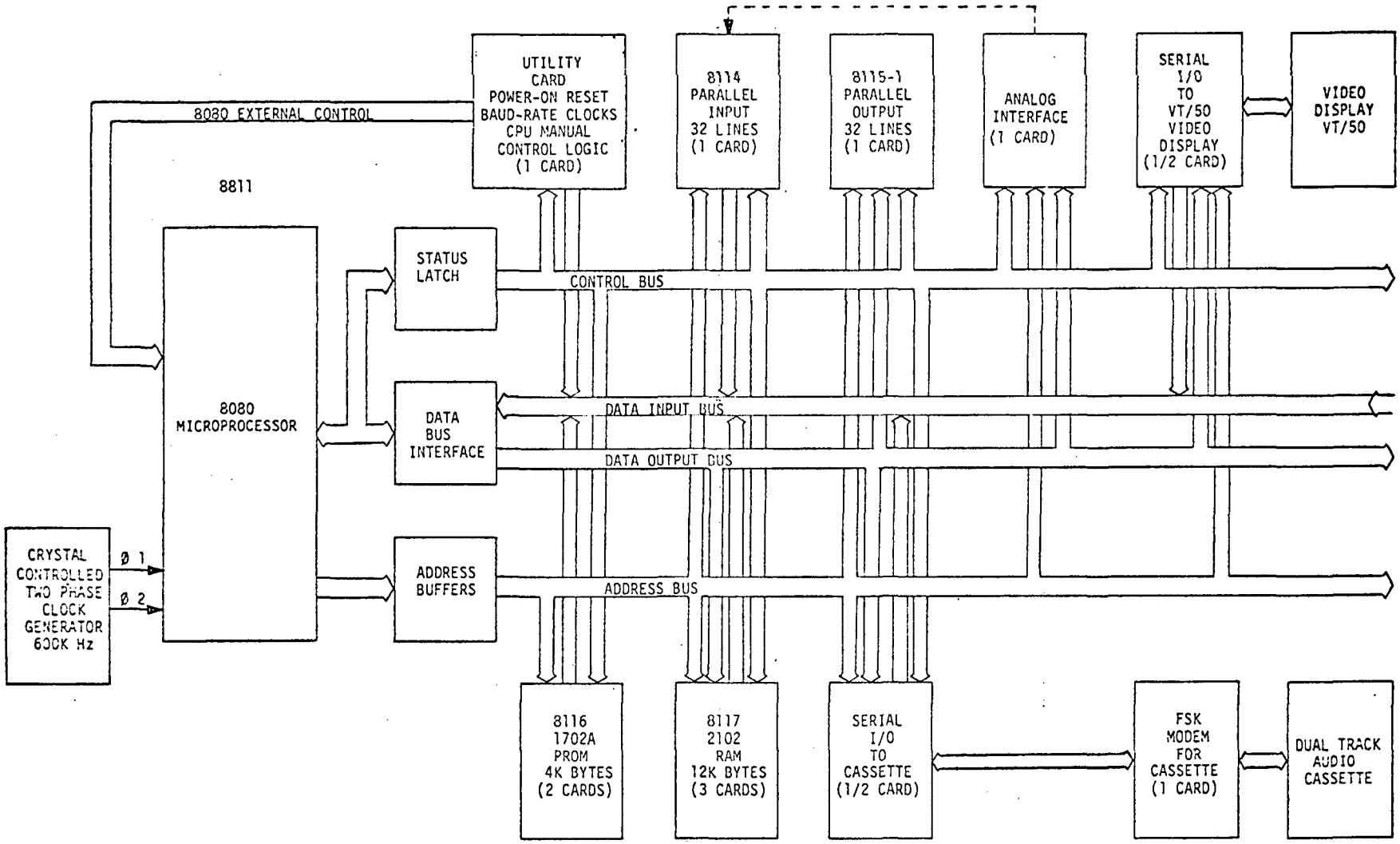


Figure 5.1(2) UT PDCP Microcomputer Development System Block Diagram.

The 8080 based Pro-Log Corporation Microprocessor System (MPS) card components offer a stripped-down, cost-effective basic system. With the addition of four specially designed cards, a powerful PDCP development system is obtained. Other companies were interviewed as a source of basic system hardware; however, on the basis of economics and availability, a decision was made to use a set of eight basic cards made by Pro-Log Corporation and add to them four specially designed cards to create a special purpose microcomputer. Perhaps a better PDCP development system could have been designed starting with only a microprocessor chip set and no commercial cards. However, due to the short time span of this study, the ideal approach had to be compromised. Thus, the Pro-Log MPS cards provide the basic microcomputer system functions, and four add-on cards were designed to form a powerful PDCP development system.

5.1.1.1 Central Processing Unit Card - The major component of the Pro-Log system is the 8811 Central Processing Unit (CPU) card. The CPU card contains an Intel 8080 microprocessor, a two-phase clock, a power-up reset circuit, and data, address, memory, and I/O control logic. A few timing signals needed for special functions were not brought out to the edge connector on the stock CPU card. Thus, these signals were either brought out on spare edge connector pins or generated externally on the utility card (see Section 5.1.3). To facilitate accurate timing for PDCP transmitter demonstration programs, the crystal in the clock circuit for the CPU was changed from 5.0 MHz to 4.8 MHz. The 4.8 MHz clock provides a basic $1.6\overline{66}$ microsecond (μsec) state time for all instructions. This state time was chosen to provide an integer number of states between phases of a Manchester encoded data stream transmitted at the typical DCP rates (100, 400, or 5000 bits per second). Note that a $1.6\overline{66}$ μsec state time is over three times slower than the nominal 500 nanosecond (nsec) state time for a full speed 8080 system. The slower state time also permits the use of low speed, inexpensive memory without the need for wait-cycle generation circuitry.

5.1.1.2 Random-Access Memory Cards - There are three 8117 Random-Access Memory (RAM) cards in the basic Pro-Log system which

provide a total of 12K of RAM. Each RAM card provides 4K by eight bits of statis RAM (2102 type, 1.0 μsec access time). An actual PDCP would not require such a large amount of RAM. The UT PDCP is provided with 12K bytes of RAM to facilitate program development and to provide a storage area for the text of the extensive comments which accompany the PDCP demonstration software. The 12K of RAM permits all the programs of the PDCP demonstration package to be resident simultaneously with room for additional programs.

Although the access time of the 2102 type RAM's used on the three 8117 RAM cards is faster (1.0 μsec compared to $1.6\overline{66}$ μsec state time) than the basic state time of the CPU, the frequency of the system clock is limited by the access time of the read-only memory (ROM). A wait-cycle generator circuit could have been added to obtain maximum overall system speed; however, different cycle times for RAM, ROM, and non-memory referencing instructions would have complicated software timing routines. Maximum system speed is not needed to simulate all DCP functions performed by current DCP's. The three 8117 RAM cards are assigned absolute memory addresses 0-12K.

5.1.1.3 Read-Only Memory Cards - The two 8116 Read-Only Memory (ROM) cards provide non-volatile program storage. A powerful system monitor program resides in slightly less than 3.5K of ROM. General purpose binary math subroutines occupy the remaining 0.5K of ROM. Both ROM cards accept up to eight 256 x 8 bit Intel 1702A type PROM's. Thus, each card can accommodate up to 2K of ROM. To keep costs low and still meet maximum data rate specifications, 1.7 μsec maximum access time 1702A ROM's are used.

The ROM's are ultraviolet-eraseable memories and are programmed with the department's Intellec 8/Mod 80 microcomputer. The two 8116 ROM cards have been assigned the 16-20K absolute memory addresses.

5.1.1.4 Latched Parallel Output Card - Thirty-two parallel latched output lines are provided by a Pro-Log 8115-1 Output Card. The parallel output card is organized as four 8-bit parallel output ports. Execution of an "OUT" instruction causes the eight bits in the accumulator to be

latched into a particular eight-bit output port depending on the port address specified by the "OUT" instruction. The output lines are TTL compatible with a fanout of 10. The four 8-bit parallel output ports are assigned the absolute output port addresses Ø-3. Output port Ø is normally used for PDCP data output. Bit Ø of port Ø is normally a serial data output for PDCP transmitter formatted data. Other bits of port Ø provide scope data and synchronization signals as well as front panel LED data displays. Output port 1 is normally used to drive PDCP status displays and PDCP transmitter control signals. For example, four bits are used to drive the front panel PDCP mode displays (Landsat, GOES, TIROS-N, or TWERLE mode). Other bits of output port 1 represent signals to control platform functions such as platform power on and off, RF power on and off, data enable, and data integrator initial conditions control. Presently, output port 2 controls the cassette read/write flag LED, while port 3 is brought out to the rear panel for transmitter timing verification.

An additional parallel output port is provided on the dual serial I/O card (see Section 5.1.2.1) to control the cassette recorder interface.

5.1.1.5 Parallel Input Card - The last member of the Pro-Log MPS components card family is the 8113, 32-line parallel input card. Like the parallel output card, the input card is organized as four 8-bit parallel ports. The parallel input card provides parallel data input. The input ports are assigned absolute addresses Ø-3, the same as the parallel output ports. The eight-bit data switch register on the front panel is connected to input port Ø.

The switch register serves as simulated parallel sensor data. The eight-bit byte loaded on the data switch register is input to the accumulator of the CPU upon execution of the "IN Ø" instruction. The remaining three parallel input ports provide additional simulated sensor. inputs.

The dual serial I/O card also has a parallel input port (see Section 5.1.2.1) which is used to input status of the cassette recorder interface, the digital-to-analog converter, and the two Universal Asynchronous

Receiver Transmitter (UART) chips used to interface the serial data of the video display terminal and the cassette recorder.

5.1.2 Special Purpose Interface Cards

The eight 8080 based Pro-Log MPS components cards provide the basic functions of a microcomputer: CPU, memory, and parallel input/output. Four special-purpose interface cards have been added to the basic Pro-Log system to create a powerful special-purpose microcomputer system capable of simulating all the tasks performed by present DCP designs [see Figure 5.1(2)].

The dual serial I/O card was developed to interface the serial data format of the video display terminal to the parallel bus structure of the CPU. The second serial interface provides serial/parallel conversions for the cassette recorder interface to and from the CPU bus. A parallel input and output port is included on the dual serial I/O card to input status from the serial interfaces and cassette recorder and to output control signals to the cassette interface card and the external frequency counter.

The analog interface card provides simulated sensor input of analog data. The card contains an eight-bit digital-to-analog (DAC) converter, a comparator, a voltage-to-frequency converter and the necessary handshake logic to permit software analog-to-digital conversion.

The cassette interface card uses a frequency shift keyed (FSK) technique to store and retrieve data from an inexpensive cassette recorder. A subharmonic of the baud rate clock used to clock data to the cassette is also recorded. Thus, on playback, the subharmonic of the baud rate clock is used to phaselock the baud rate clock decoding the data. This technique permits large speed fluctuations from the cassette with no effect on the error rate of the data.

The general purpose utility card provides several special functions including hardware system start-up; baud rate clock and current loop interface for the video terminal; and single step control logic. A more detailed description of the four special purpose cards is provided below.

5.1.2.1 Dual Serial I/O Card - The dual serial I/O card and the other three special-purpose cards were designed to be directly compatible with the address and data-bus structure of the Pro-Log 8811 CPU card. The bus inputs and outputs are buffered so that the CPU bus driving capability is preserved. Two serial input and two serial output ports are implemented on the dual serial I/O card as well as a parallel input and output port.

The major component of the dual serial I/O card is a MOS, LSI device called a Universal Asynchronous Receiver Transmitter (UART). The UART performs serial-to-parallel and parallel-to-serial data conversion as well as providing status information on the states of the input and output buffers included in the device. The UART can be programmed to receive and transmit five to eight bit words, with or without parity and with one or two stop bits. The UART interfacing the video terminal is hardwire programmed for an eight-bit word, one stop bit and no parity. The second UART interfaces the serial data format of the cassette recorder and is programmed under software control via the parallel output port included on the dual serial I/O card.

The parallel output port also controls the motor of the cassette deck. Also provided on the dual serial I/O card is an eight-bit parallel input port used to input UART status. Port absolute-address decoding is included on the card, making it a stand-alone card except for baud rate clocks needed to drive the UART's. A standard Pro-Log single serial interface card would have required connection to the Pro-Log parallel input and output cards due to inadequate buffering of their serial interface card and the lack of address decoding. The specially designed dual serial I/O card not only provides two serial I/O ports and all the CPU handshaking required, but it is independent of the parallel I/O cards thus freeing all 64 parallel I/O lines for special purpose use.

The serial I/O data absolute port address assignment for the video display terminal is port 7. Input status and output cassette control and UART programming has been assigned to port 6.

5.1.2.2 Analog Interface Card - The analog interface card contains an eight-bit CMOS digital-to-analog converter (DAC), a comparator, a voltage-to-frequency converter, and CPU handshake logic. The eight-bit DAC and comparator are used in a software successive approximation analog-to-digital conversion technique. The voltage-to-frequency converter provides a second, less expensive technique for analog-to-digital conversion. The frequency output is counted with a software frequency counter routine resulting in a digital conversion of the analog input. The software routines for the successive approximation analog-to-digital conversion (ADC) and the voltage-to-frequency to digital conversion are included in PROM.

5.1.2.3 Cassette Analog Interface and Baud Rate Clock Card - The audio interface to and from the cassette recorder is provided by the cassette analog interface and baud rate clock card. Serial data from the cassette serial data output port of the dual serial I/O card is converted to AFSK tones of 4,500-Hz (space tone) and 5,500-Hz (mark tone). The AFSK signal is created on the cassette analog card and is recorded on the right channel of the cassette recorder. In playback, the recovered AFSK tones from the right channel are converted back to serial TTL compatible levels with an EXAR-210 FSK demodulator integrated circuit. The recovered serial data from the cassette analog card is then applied to the dual serial I/O card's cassette serial input port.

Simultaneous recording of a 600-Hz tone on the left channel is provided by a baud rate timing circuit on the cassette analog and baud rate timing card. A master clock frequency of 57,600-Hz is divided down to 600-Hz. The divider chain provides optional baud rates of 300, 600, 1200, and 1800 baud. The 1200 baud clock was selected for reliable, fast data recording.

The master 57,600-Hz clock is derived from two sources. In record, a 555 Timer IC operating as a fixed frequency astable oscillator supplies

the master clock frequency. In playback, the recovered 600-Hz subharmonic tone is used to phase-lock a VCO to 96 times the recovered 600-Hz tone. The synthesized recovered baud rate clock provides timing for the serial decoding of recovered data. Since the recovered baud rate timing suffers approximately the same timing errors as the recovered data, overall serial data decoding is relatively immune to timing errors due to the speed variations of an inexpensive cassette recorder.

5.1.2.4 General-Purpose Utility Card - This card contains:

1. Hardware monitor start-up circuit.
2. Baud rate clock for the video display terminal.
3. TTY current interface for the video display terminal.
4. Single step and other control logic.

A single push button switch is used to activate a hardware start-up circuit. In addition, a power-up circuit on the CPU card also activates the start-up circuit so that application of power starts the system.

The dual serial I/O card provides TTL logic level serial data. The utility card interfaces the standard 20mA current loop of the video display terminal to the TTL logic levels of the dual serial I/O card. A baud rate clock for the video terminal is included on the utility card providing switch selectable standard baud rates.

Synchronous run, wait, and single step logic is provided for hardware control of the UT PDCP. In addition, a special "slow-run" circuit is provided to permit slow execution of a program. This feature is particularly useful for demonstrating PDCP transmitter routines.

5.1.3 Miscellaneous Hardware Components of the UT PDCP Development System

The eight Pro-Log MPS components cards and the four special purpose interface cards plug into a small 12.7cm x 25.4cm (5" x 10") rack with

wire-wrapped edge connectors. The cards are powered by three voltages (+5, +12, and -9 volts) obtained from two commercial short-circuit proof power supply modules which were made specifically for the Pro-Log MPS components cards. The analog interface card also utilizes a -12 volt power supply which was added to the system.

An attractive gray cabinet houses the small rack containing the 12 system cards. The AC power supply is external and connected via a long flexible cable. The front panel of the cabinet contains a minimum number of switches and displays in an effort to simplify use of the UT PDCP system. A set of eight switches provides a fixed value data input source and is useful in simulating a sensor data source. The value toggled on the data switch register is input to the accumulator of the CPU upon execution of an "IN Ø" instruction. Other switches on the front panel include wait, single step, slow-run, start, power-on, and cassette motor-on. The function of these switches is explained in the System Operation Manual. Several LED's are included on the front panel to simulate special functions being performed by PDCP software routines. Finally, BNC-type jacks located on the front panel are used to display signals on an external oscilloscope, and one jack can be connected to an external frequency counter to provide timing verification of PDCP functions. The LED indications and signals appearing at the BNC jacks are explained in detail in the System Operation Manual.

The rear panel of the UT PDCP contains a jack for interconnection of the cassette recorder. Another connector provides interconnection to the video-display terminal. A baud rate select switch for the serial teletype interface also appears on the rear panel. The terminal baud rate switch allows connection of a hard copy teleprinter (such as an ASR-33) in place of the video terminal in the event hard copy is desired.

User interaction with the UT PDCP system is provided by a Digital Equipment Corporation VT-50/CA video diesplay and data entry terminal. The VT-50/CA communicates via a full-duplex serial TTY current loop. A TTY current loop interface is included on the general-purpose utility card. Full duplex operation is maintained to achieve maximum versatility of the VT-50/CA. The VT-50/CA provides an 80-character per line, 12 line

alpha-numeric display and a multimode ASCII two-key rollover keyboard.

The connector feeding the VT-50/CA could be used to operate any other ASCII TTY 20mA current-loop device such as the industry standard low-speed ASR-33. Most applications for the UT PDCP system are best suited to a high-speed video terminal with cursor control like the DEC VT-50/CA; however, hard copy of program listings and data can be obtained using an ASR-33 teletype if desired.

Several cassette recorders were tested for suitability as a bulk program storage device. Of all the recorders tested, the Lafayette RK-725 stereo cassette deck exhibited the best frequency response (approximately twice the frequency response of all other units tested). Since a speaker and high-power audio amplifier are not needed, a cassette deck is preferable to a cassette player. The high-frequency data audio tones for the cassette interface take advantage of the high-frequency response of the Lafayette RK-725. Higher frequency AFSK tones reduce jitter from the phase locked loop demodulator. Also, having two channels permits excellent isolation between the AFSK tones and the baud rate subharmonic tone. No filter is required to separate the tones as would be the case for a monaural recorder. Frequency jitter on playback (a common problem of inexpensive recorders) was not substantial and as discussed in Section 5.1.2.3, a 600-Hz baud rate subharmonic tone recording is used to eliminate timing errors in decoding the data.

In summary, the hardware for the UT PDCP development system consists of a special purpose 8080-based microcomputer, a DEC VT-50/CA video terminal, and a cassette recorder and interface for program storage.

5.2 UT PDCP DEVELOPMENT SYSTEM SOFTWARE PACKAGE

To develop a large number of PDCP demonstration programs in the short time period of this study, software support must be extensive. At the beginning of this research period, the only microprocessor family possessing substantial software support was the Intel 8080 microprocessor

family. Since then, several companies have second-sourced the 8080 including improved versions. Several software firms have also added software support. Intel's user's library contains many useful programs, including a Macro Cross Assembler which runs on the DEC PDP-11 minicomputer. Dr. Steve Olsen of the University of Utah supplied a paper tape of his cross assembler, and all of the PDCP programs written during the research period were assembled on the department's PDP-11/40 using a modified version of Dr. Olsen's cross assembler.

Microcomputer software is often developed using the hexadecimal or octal number systems. The PDCP programs developed on the PDP-11/40 use an octal format whereas standard Intel 8080 software is provided in a hexadecimal format. In addition, non-programmers may not be familiar with either hexadecimal or octal and, thus, would prefer to work in decimal. To facilitate the development of PDCP programs a multiradix system monitor was developed for the UT PDCP development system. This monitor will execute commands in octal, hexadecimal, or decimal. For example, programs can be listed in any of the three radices.

The PDCP resident system monitor provides several key functions for the operation of the UT PDCP system. Examination and modification of memory contents, cassette storage and retrieval of bulk data, radix conversions, and program execution commands with debugging techniques are among the many system monitor functions which have been incorporated under this study. In addition, a special keyboard assembler command permits programming of the UT PDCP directly from the keyboard using standard Intel instruction mnemonics. Finally, memory contents may be displayed symbolically using a special list symbolic command. The PDCP resident monitor resides in non-volatile ROM and is available immediately on system start-up. The two 8116 Pro-Log PROM cards contain the entire PDCP system monitor. The System Operation Manual support software section describes in detail the resident monitor software and commands.

The system monitor requires about 3.5K of the available 4K of PROM. The remaining 0.5K of PROM contains general purpose PDCP subroutines. Having these often used subroutines resident in non-volatile memory

simplifies source program writing as the routines are simply called when needed instead of repeating the source code of the entire subroutine. Source listings and explanations of these PDCP subroutines are described in the System Operation Manual support software section.

6. FUTURE PDCP SYSTEMS

The last five years have witnessed the birth, development, and application of the microprocessor. It is difficult to accurately predict what will happen during the next five years in this field. Semiconductor manufacturers are already working on third-generation microprocessors. The designers of data collection system platforms should have available to them a prediction or reasonable projection of the future characteristics and capabilities of the microprocessor if a programmable data collection platform is to be considered. The purpose of this section of the report is to provide a method of microprocessor selection and to furnish the designer with a microprocessor capability forecast for the next five-year period.

6.1 EVALUATION OF AVAILABLE MICROPROCESSORS

The research proposal for this contract projected the source-destination matrix [1] for evaluation of microprocessor instruction sets. In this technique, an instruction is considered as the transfer of data from a selected source to a selected destination. By constructing a matrix in which the sources are in rows and the destination in columns, the instruction set and functional operations of the microprocessor can be depicted in a concise format. However, the source-destination matrix has two significant disadvantages. First, no accepted methods exist for evaluating or establishing a performance measurement for the microprocessor from the source-destination matrix such that a comparison of different microprocessors can be made. Secondly, the source-destination matrix does not integrate other system characteristics into the evaluation.

Present trends in system evaluation are toward the development of classification and performance measures which integrate all the operating characteristics of a system. For a microprocessor system, this includes system constraints, hardware organization, memory hierarchy, software structures, and application directorates. In fact, most of these considerations can be represented by the following functional form:

$$\text{Performance Measure} \overset{\Delta}{=} f(\text{system constraints, application constraints, software constraints}) .$$

System constraints include power, weight, size, and speed considerations, while application constraints focus on the specific computational tasks to be executed. Thus, application constraints more clearly define the capability of various microprocessors to meet computational requirements. Software constraints are imposed by the failure to provide software support for the system. For instance, new applications are difficult to program and implement for a microprocessor system with no cross assembler, editor, or simulator.

This research has directed effort toward generating performance measures relating to system, application, and software constraints. A systematic procedure for generating a performance measure is implemented by constructing a system for matrices relating these constraints to the different microprocessors. Consider the following matrix form:

Constraints	Importance Weighting Matrix	μP_1	....	μP_n
System	$\overline{W}_1$	$\overline{R}_{11}$	....	$\overline{R}_{n1}$
Application	$\overline{W}_2$	$\overline{R}_{12}$	....	$\overline{R}_{n2}$
Software	$\overline{W}_3$	$\overline{R}_{13}$	....	$\overline{R}_{n3}$

$\overline{W}_i$ is a weighting matrix associated with the importance of a particular constraint; whereas, $\overline{R}_{ji}$ is a rating or measure matrix for μP_j (the jth microprocessor) which evaluates the capability of the microprocessor to satisfy the conditions imposed by the constraint. The performance measure, PM_j, for μP_j is given by

$$PM_j \triangleq [\bar{W}]' [\bar{R}_j]$$

$$= [\bar{W}_1 \ \bar{W}_2 \ \bar{W}_3]' \begin{bmatrix} \bar{R}_{j1} \\ \bar{R}_{j2} \\ \bar{R}_{j3} \end{bmatrix}$$

$$= \sum_{i=1}^{3} \bar{W}_i \bar{R}_{ji} .$$

Within each general constraint there are numerous conditons for which the measure matrix, $\bar{R}_{ji}$, must be established. For instance, consider the submatrix formed for the general system constraints in Table 6.1(1). A weighting factor, W_{1i}, is associated with the importance of the ith parameter. In evaluating a given microprocessor, a measure (r_{j1i}) of the ability of the microporcessor to satisfy the parameter requirement can be established. Similar submatrices are formed for the applications constraints of Table 6.1(2) and the software constraints of Table 6.1(3).

To provide an example of this method for microprocessor evaluation, a comparsion of the INTEL 8080A and RCA COSMAC microprocessors is developed. In generating each measure matrix, the difference in performance measures is the dominant consideration. The absolute values of the measures would be adjusted as additional information about other microprocessor families is included. Each measure value is established on a scale from 1 to 100. The larger measure values indicate the microprocessor to be more suitable in satisfying the parameter constraints.

If a large number of microprocessors are included in the evaluation, the system lends itself to computer implementation for bookkeeping purposes. A computer with higher-order language capability and containing matrix multiplication features such as APL is ideally suited for the bookkeeping task. Changes in absolute value of the measures can be easily entered and the performance measures can be quickly recomputed.

TABLE 6.1(1)

SYSTEM CONSTRAINTS

Parameter	Weighting Matrix $\overline{W}_1$	Measure Matrix $\overline{R}_{j1}$
Power Consumption	W_{11}	r_{j11}
Number of Power Supplies	W_{12}	r_{j12}
Weight	W_{13}	r_{j13}
Size	W_{14}	r_{j14}
Cost	W_{15}	r_{j15}
Minumum System Complexity	W_{16}	r_{j16}
Availability of MSI and LSI Support Logic	W_{17}	r_{j17}
Second Sourcing	W_{18}	r_{j18}

TABLE 6.1(2)

APPLICATION CONSTRAINTS

Parameter	Weighting Matrix $\overline{W}_2$	Measure Matrix $\overline{R}_{j2}$
INSTRUCTION SET		
Arithmetic Inst.		
ADD, SUB, AND, EX-OR, OR	W_{21}	r_{j21}
Multiply Divide	W_{22}	r_{j22}
Addressing Modes	W_{23}	r_{j23}
Subroutine Linkage	W_{24}	r_{j24}
Bit Manipulation	W_{25}	r_{j25}
I/O Operations	W_{26}	r_{j26}
Conditional Inst.	W_{27}	r_{j27}
MICROPROCESSOR ORGANIZATION		
Accumulators, Working Registers	W_{28}	r_{j28}
Hardware, Software Stack	W_{29}	r_{j29}
Control	W_{2A}	r_{j2A}
Interrupt Structure Software, Trap Vector	W_{2B}	r_{j2B}
BENCH MARK PROGRAMS		
Program Storage	W_{2C}	r_{j2C}
Execution Time	W_{2D}	r_{j2D}

TABLE 6.1(3)

SOFTWARE CONSTRAINTS

Parameter	Weighting Matrix $\overline{W}_3$	Measure Matrix $\overline{R}_{j3}$
Resident Assembler	W_{31}	r_{j31}
Resident Editor	W_{32}	r_{j32}
Cross Assembler	W_{33}	r_{j33}
Simulator	W_{34}	r_{j34}
PL-1	W_{35}	r_{j35}
FORTRAN	W_{36}	r_{j36}
BASIC	W_{37}	r_{j37}
Other High-Level Languages	W_{38}	r_{j38}

The system measure matrices, $\overline{R}_{I1}$ and $\overline{R}_{R1}$, from Table 6.1(1) for the INTEL 8080A and the RCA COSMAC is determined to be

$$\overline{R}_{I1} = \begin{bmatrix} r_{I11} \\ r_{I12} \\ r_{I13} \\ r_{I14} \\ r_{I15} \\ r_{I16} \\ r_{I17} \\ r_{I18} \end{bmatrix} = \begin{bmatrix} 20 \\ 40 \\ 40 \\ 40 \\ 60 \\ 50 \\ 75 \\ 90 \end{bmatrix} \quad \text{and} \quad \overline{R}_{R1} = \begin{bmatrix} r_{R11} \\ r_{R12} \\ r_{R13} \\ r_{R14} \\ r_{R15} \\ r_{R16} \\ r_{R17} \\ r_{R18} \end{bmatrix} = \begin{bmatrix} 95 \\ 100 \\ 90 \\ 60 \\ 55 \\ 80 \\ 50 \\ 50 \end{bmatrix}.$$

In an attempt to justify a few of the measure values, the following comments are provided. Notice that $r_{I11} = 20$ and $r_{R11} = 95$. The justification of this difference is simple. COSMAC consumes an order of magnitude less power than the 8080A. Next, $r_{I12} = 40$ and $r_{R12} = 100$. The 8080A requires three power supplies while COSMAC requires only one unregulated supply. On the other hand, $r_{I17} = 75$ and $r_{R17} = 50$. INTEL provides support hardware in a family of integrated circuits which is superior to that provided by RCA at this time. There are similar considerations for establishing each measure value in $\overline{R}_{I1}$ and $\overline{R}_{R1}$.

Next, consider the system constraints performance measure, $SPM_j = \overline{W}_1' \overline{R}_{j1}$. If the weighting matrix, $\overline{W}_1$, is chosen as unity,

$$SPM_I = \frac{\overline{W}_1' \overline{R}_{I1}}{\sum W_{1n}} = 60.00$$

and

$$SPM_R = \frac{\overline{W}_1' \ \overline{R}_{R1}}{\sum W_{1n}} = 82.86,$$

where the product $\overline{W}_1' \ R_{ji}$ has been normalized to a value of 100 for the ideal microprocessor by dividing the result by the sum of the system constraint weights, $\sum W_{1n}$.

COSMAC makes a somewhat better showing; however, a more realistic weighting matrix for DCP applications is

$$\overline{W}_1 = \begin{bmatrix} W_{11} \\ W_{12} \\ W_{13} \\ W_{14} \\ W_{15} \\ W_{16} \\ W_{17} \\ W_{18} \end{bmatrix} = \begin{bmatrix} 100 \\ 5 \\ 3 \\ 1 \\ 10 \\ 8 \\ 8 \\ 15 \end{bmatrix}.$$

W_{11} is weighted more heavily than all other factors combined since power consumption is the most important consideration. Among the remaining system constraints, the availability of second sources for the microprocessor is the most important since this provides some protection against long delivery delays and premature removal of the product from the market. $W_{15} = 10$, indicating cost is not as important when comparing microprocessor-based systems. This weight could become increasingly more important if a microprocessor-based PDCP was being compared to a hardwired DCP.

Using the more realistic choice of weights for the weighting matrix, the system performance measures become

$$SPM_I = 35.90$$

and

$$SPM_R = 84.47 .$$

The superiority of COSMAC to satisfy system constraints in PDCP applications becomes more evident.

Generating the measure matrix, $\overline{R}_{j2}$, for the application constraints of Table 6.1(2) yields the following two matrices:

$$\overline{R}_{I2} = \begin{bmatrix} r_{I21} \\ r_{I22} \\ r_{I23} \\ r_{I24} \\ r_{I25} \\ r_{I26} \\ r_{I27} \\ r_{I28} \\ r_{I29} \\ r_{I2A} \\ r_{I2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 0 \\ 70 \\ 95 \\ 40 \\ 50 \\ 90 \\ 80 \\ 80 \\ 80 \\ 90 \end{bmatrix} \quad \text{and} \quad \overline{R}_{R2} = \begin{bmatrix} r_{R21} \\ r_{R22} \\ r_{R23} \\ r_{R24} \\ r_{R25} \\ r_{R26} \\ r_{R27} \\ r_{R28} \\ r_{R29} \\ r_{R2A} \\ r_{R2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 0 \\ 70 \\ 60 \\ 35 \\ 80 \\ 75 \\ 65 \\ 40 \\ 80 \\ 50 \end{bmatrix}$$

The measures relating to bench mark programs are not included since programs for COSMAC have not been sufficiently developed to provide a realistic comparison. Note that $r_{I22} = 0 = r_{R22}$. Neither microprocessor provides a hardware multiply or divide at the present time. Since the 8080A has superior subroutine linkage, $r_{I24} = 95$ and $r_{R24} = 60$. On the other hand, COSMAC provides for better I/O communications; therefore, $r_{I26} = 50$ and $r_{R26} = 80$. The stack operations of the Intel 8080A are far superior to those of COSMAC, resulting in $r_{I29} = 80$ and $r_{R29} = 40$.

Considering the importance of the measure in the application constraints of PDCPs, the weighting matrix, $\overline{W}_2$, is determined to be

$$\overline{W}_2 = \begin{bmatrix} W_{21} \\ W_{22} \\ W_{23} \\ W_{24} \\ W_{25} \\ W_{26} \\ W_{27} \\ W_{28} \\ W_{29} \\ W_{2A} \\ W_{2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 20 \\ 75 \\ 75 \\ 75 \\ 65 \\ 55 \\ 50 \\ 60 \\ 60 \\ 20 \end{bmatrix} .$$

The comparison of the application constraints performance measure, $APM_j = \frac{\overline{W}_2' \overline{R}_{j2}}{\sum W_{2n}}$, for the two microprocessor yields

$$APM_I = \frac{\overline{W}_2' \overline{R}_{I2}}{\sum W_{2n}} = 74.77$$

and

$$APM_R = \frac{\overline{W}_2' \overline{R}_{R2}}{\sum W_{2n}} = 65.88 .$$

These results indicate that the Intel 8080A is a better choice in satisfying application constraints.

Finally, the measure matrices for the software constraints of Table 6.1(3) are determined.

$$\overline{R}_{I3} = \begin{bmatrix} r_{I31} \\ r_{I32} \\ r_{I33} \\ r_{I34} \\ r_{I35} \\ r_{I36} \\ r_{I37} \\ r_{I38} \end{bmatrix} = \begin{bmatrix} 90 \\ 90 \\ 90 \\ 90 \\ 60 \\ 50 \\ 90 \\ 50 \end{bmatrix}$$

and

$$\overline{R}_{R3} = \begin{bmatrix} r_{R31} \\ r_{R32} \\ r_{R33} \\ r_{R34} \\ r_{R35} \\ r_{R36} \\ r_{R37} \\ r_{R38} \end{bmatrix} = \begin{bmatrix} 80 \\ 80 \\ 95 \\ 90 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

At the present time, INTEL provides better software support than RCA, especially in compiler design. However, compiler design is less important in PDCP applications than the availability of a cross assembler and simulator programs. Thus, the weighting matrix, $\overline{W}_3$, for software constraints is determined to be

$$\overline{W}_3 = \begin{bmatrix} W_{31} \\ W_{32} \\ W_{33} \\ W_{34} \\ W_{35} \\ W_{36} \\ W_{37} \\ W_{38} \end{bmatrix} = \begin{bmatrix} 75 \\ 75 \\ 90 \\ 80 \\ 30 \\ 10 \\ 0 \\ 1 \end{bmatrix} .$$

The software constraints performance measure for the two microprocessors is

$$SOPM_I = \frac{\overline{W}_3' \, \overline{R}_{I3}}{\sum W_{3n}} = 86.29$$

and

$$SOPM_R = \frac{\overline{W}_3' \, \overline{R}_{R3}}{\sum W_{3n}} = 76.87 \ .$$

The software support performance measure of the 8080A is not really that superior to COSMAC.

Linearly combining the results generated from these considerations, the overall performance measure for the two microprocessors is evaluated to be

$$PM_I = \frac{\overline{W}' \, \overline{R}_I}{\sum W_i} = 65.65$$

and

$$PM_R = \frac{\overline{W}' \overline{R}_R}{\sum W_i} = 75.74 .$$

The RCA 1802's performance measure is only ten percentage points higher than the INTEL 8080's. However, in PDCP applications, system constraints will generally be considerably more important than either application or software constraints. A typical importance weighting matrix for the PDCP application is

$$\overline{W} = \begin{bmatrix} 100 \\ 40 \\ 20 \end{bmatrix}$$

The overall performance measures for the 8080A and 1802 microprocessors are

$$PM_I = 51.92$$

and

$$PM_R = 78.87 .$$

The 1802 now has a clear performance advantage of nearly 27 percentage points.

The attempt here is not to conclusively state which of the available microprocessors is better suited for a microprocessor-based PDCP, but rather, to emphasize this technique as a method for evaluating and comparing microprocessors for suitability in PDCP applications. Performance measures derived using this technique are primarily intended to aid in choosing between two or more microprocessors which are known to meet the basic constraints of the application. Invalid results can be obtained if one attempts to make an "apples to oranges" type comparison by failing to eliminate machines with unacceptable characteristics before proceeding with the performance evaluation. For example, a bipolar

microprocessor should be eliminated on the basis of unacceptable power consumption even though program execution time would be much shorter than for a CMOS or NMOS microprocessor. This limitation of the performance evaluation procedure is not a significant handicap since the initial screening process is relatively easy to accomplish. The final screening, which is much more difficult to perform, can be simplified by applying the performance evaluation procedure developed in this section.

6.2 TECHNOLOGY FORECAST

In selecting whether to include the programmable feature in a data collection platform system, the designer should have available any projections which indicate the characteristics and capabilities of semiconductor devices for a period covering the next five to ten years. While these predictions are difficult to obtain, reasonably accurate models can be developed which yield satisfactory results over this time span.

There are a number of methods of technology forecasting [2] which will lead to the development of a set of performance characteristics of components during a particular period in the future. These different methods can be reduced to two basic classes of forecasting techniques. One method, called trend forecasting, involves the development of a mathematical model which utilizes past history as a guide for the projection in the future. The second utilizes the judgement of experts in the field to predict the changes in future technology. This is termed intuitive forecasting.

There are essentially two components which will have the greatest effect upon the design of a programmable data collection system. One component is the microprocessor chip itself, and the other is the memory associated with the microprocessor. Already microprocessors are beginning to appear with clock, input/output buffers and control logic on a single chip. There is even some attempt to include a small memory area on the microprocessor substate making a true single-chip microcomputer.

Trend forecasting has predicted that the average computer add time will decrease by one order of magnitude in a decade [3]. Five years from now, one could expect the average computer add time to be one-third that of the present day average add time. The same performance improvement can be expected in microprocessors. The average present day microprocessor has an add time of two microseconds. In five years, the average add time is expected to be 670 nanoseconds.

Semiconductor memories have taken a more dramatic reduction in chip area, power, and cost requirements. The area of the memory cell is decreasing at the rate of one order of magnitude in seven years. Some experts believe that a 128K-bit memory chip could be available in 1980. Therefore, it seems reasonable to predict that a microprocessor with a 600-nanosecond add time and 8K of eight-bit words could be fabricated on a single chip within five years. The cost of such a device will be in the $25 range. With powerful computational capabilities such as these becoming available in the not too distant future, the programmable data collection platform should be a viable element in data collection systems.

The most promising low power technologies for use on future PDCPs appear to be silicon-on-sapphire CMOS (SOS), closed cell COS/MOS logic (C^2L), and integrated injection logic (I^2L). I^2L is a bipolar circuit design technique which significantly increases the density of bipolar circuits and permits operation at any point along a constant speed-power product line spanning several decades of propagation delay and power consumption. Thus, low power operation can be achieved at the cost of increased propagation delay or speed can be improved if higher power consumption is acceptable. Although future production of low power microprocessor systems using I^2L is feasible, most manufacturers are currently concentrating I^2L development efforts in the areas of linear-to-digital interface circuits and combinations of linear and digital processing on a single chip. Many I^2L designs use a level conversion circuit to provide a simple interface between the I^2L I/O signals and standard TTL logic. A CMOS to I^2L interface should also be feasible since most new CMOS designs are capable of driving a TTL load. Thus,

I^2L circuits may provide LSI combinations of digital and analog peripheral functions for future microprocessor based PDCPs.

The standard bulk CMOS process has recently been improved by two different techniques which both have excellent potential for providing future advances in microprocessor systems. Commercial products using SOS and C^2L technologies are already available. SOS integrated circuits are manufactured using many of the standard bulk CMOS processing procedures. The main distinction between SOS and bulk CMOS is that a sapphire substrate is utilized in SOS processing to reduce load capacitance and improve circuit density. Propagation delay is also reduced. High speed, low power, SOS memories which are compatible with the 1802 microprocessor are currently manufactured by RCA. These devices exhibit lower power consumption than bulk CMOS devices for operating speeds above approximately 1KHz but consume more power than bulk CMOS devices in static operation.

The most recent advancement in CMOS processing is the closed cell COS/MOS, or C^2L, process developed by RCA for the 1802 microprocessor. C^2L is a circuit design technique that permits a common source structure for 200 to 300 transistors. The transistor's gate electrode forms a closed circle which provides gate termination and eliminates the need for guardbands. As a result, circuit densities are approximately the same as for SOS. C^2L also results in a higher transconductance-to-drain capacitance ratio. In conjunction with a self-aligned silicon gate which reduces Miller capacitance, C^2L offers significant speed improvements over bulk CMOS. Additional refinements in the C^2L and SOS technologies are expected to provide continued improvements in microprocessors and associated devices which exhibit the low power consumption required by the PDCP application.

REFERENCES

1. Nichols, J., "The Source-Destination Matrix for Instruction Set Presentation," IEEE Spring Computer Conference, San Francisco, California, February 25-26, 1975, pp. 61-63.

2. Bright, J. R., Editor, Technological Forecasting for Industry and Government, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 146-147, 1968.

3. Wintz, et al., "Study of On-Board Processing for Earth Resources Data," Wintek Corp., Lafayette, Indiana, NASA - Ames Contract No. NASA 8327, September 1975.

APPENDIX A

BINARY MATH PACKAGE

```
;                SHORT MULTI-PRECISION ADD SUBROUTINE                  ;
;                                                                      ;
;  THIS SUBROUTINE PERFORMS BINARY ADDITION ON TWO UNSIGNED            ;
;  BINARY NUMBERS.  THE 16 BIT NUMBER IN REGISTER PAIR BC IS           ;
;  ADDED TO THE 32 BIT NUMBER STORED IN MEMORY LEAST SIGNIFI-          ;
;  CANT BYTE FIRST BEGINNING AT THE LOCATION SPECIFIED BY THE          ;
;  CONTENTS OF HL.  THE 32 BIT SUM IS RETURNED LEAST SIGNIFI-          ;
;  CANT BYTE FIRST BEGINNING IN THE MEMORY LOCATION ADDRESSED          ;
;  BY HL.  ALL MACHINE STATUS IS PRESERVED EXCEPT FOR THE ACC          ;
;  AND FLAGS.  ON RETURN, CARRY FLAG (CY) = 1 INDICATES                ;
;  AN OVERFLOW OCCURED.  THE SUBROUTINE REQUIRES 20 BYTES              ;
;  OF ROM, USES 4 BYTES OF STACK STORAGE (RAM) AND EXECUTES IN         ;
;  141 STATES.  A SHORT FORM NOTATION FOR THE PRECEDING EX-            ;
;  PLANATION IS GIVEN BELOW:                                           ;
;                                                                      ;
;  (BC) + (M<HL>) ---> M<HL>   CY=1 INDICATES OVERFLOW                 ;
;  16 BITS + 32 BITS = 32 BITS   -  ACCUMULATOR AND FLAGS              ;
;  DESTROYED.   141 STATES, 20 BYTES ROM, AND 4 BYTES STACK.           ;
000000                  SMADD:
000000                          PUSH    H               ; SAVE STATUS OF HL
000000  345
000001                          MOV     A,M             ; GET LS BYTE OF SUM
000001  176
000002                          ADD     C               ; FORM LS BYTE OF SUM
000002  201
000003                          MOV     M,A             ; STORE LEAST SIG BYTE OF SUM
000003  167
000004                          INX     H               ; INCREMENT ADDRESS POINTER
000004  043
000005                          MOV     A,M             ; GET 2ND BYTE OF SUM
000005  176
000006                          ADC     B               ; FORM SECOND BYTE OF SUM
000006  210
000007                          MOV     M,A             ; STORE SECOND BYTE OF SUM
000007  167
000010                          INX     H               ; INCREMENT ADDRESS POINTER
000010  043
000011                          MOV     A,M             ; MOVE NEXT BYTE TO A
000011  176
000012                          ACI     0               ; FORM THIRD BYTE OF SUM
000012  316     000
000014                          MOV     M,A             ; STORE THIRD BYTE OF SUM
000014  167
000015                          INX     H               ; INCREMENT ADDRESS POINTER
000015  043
000016                          MOV     A,M             ; MOVE NEXT BYTE TO A
000016  176
000017                          ACI     0               ; FORM MOST SIGNIFICANT BYTE OF SUM
000017  316     000
000021                          MOV     M,A             ; STORE MS BYTE OF SUM
000021  167
000022                          POP     H               ; RESTORE HL
000022  341
000023                          RET                     ; RETURN
000023  311
```

Program A(1) Short Multiprecision Add Subroutine.

```
;                                                                       ;
;               SHORT MULTI-PRECISION SUBTRACT SUBROUTINE               ;
;                                                                       ;
;   (M<HL>)   -   (BC)   --->   (M<HL>) CY=1 INDICATES UNDERFLOW        ;
;   32 BITS   16 BITS = 32 BITS   -   ACCUMULATOR AND FLAGS             ;
;   DESTROYED.   141 STATES, 20 BYTES ROM AND 4 BYTES STACK             ;
;                                                                       ;
000024                  SMSUB:
000024                          PUSH    H               ; SAVE HL
000024  345
000025                          MOV     A,M             ; GET LS BYTE
000025  176
000026                          SUB     C               ; SUBTRACT LS BYTE OF SUBTRAHEND
000026  221
000027                          MOV     M,A             ; STORE LS BYTE OF DIFFERENCE
000027  167
000030                          INX     H               ; INCREMENT ADDRESS POINTER
000030  043
000031                          MOV     A,M             ; GET 2ND BYTE OF MINUEND
000031  176
000032                          SBB     B               ; SUBTRACT 2ND BYTE OF SUBTRAHEND
000032  230
000033                          MOV     M,A             ; STORE PARTIAL DIFFERENCE
000033  167
000034                          INX     H               ; INCREMENT ADDRESS POINTER
000034  043
000035                          MOV     A,M             ; GET 3RD BYTE OF MINUEND
000035  176
000036                          SBI     0               ; SUBTRACT BORROW
000036  336     000
000040                          MOV     M,A             ; STORE 3RD BYTE OF DIFFERENCE
000040  167
000041                          INX     H               ; INCREMENT ADDRESS POINTER
000041  043
000042                          MOV     A,M             ; GET FOURTH BYTE OF MINUEND
000042  176
000043                          SBI     0               ; SUBTRACT BORROW
000043  336     000
000045                          MOV     M,A             ; STORE 4TH BYTE OF DIFFERENCE
000045  167
000046                          POP     H               ; RESTORE HL
000046  341
000047                          RET                     ; RETURN
000047  311
```

Program A(2) Short Multiprecision Subtract Subroutine.

```
                        DOUBLE PRECISION ADD WITH MEMORY SUBROUTINE
                ;                                                                       ;
                ;  (BC)  +  (M<HL>)  -->  M<DE>  CY=1 INDICATES OVERFLOW               ;
                ;  16 BITS + 16 BITS = 16 BITS  -  ACCUMULATOR AND FLAGS ARE           ;
                ;  DESTROYED.  63 STATES, 11 BYTES ROM, AND 2 BYTES STACK.             ;
                ;                                                                       ;
000050          DADDM:
000050                  MOV     A,M             ; GET LS BYTE FROM MEMORY
000050  176
000051                  ADD     C               ; ADD LS BYTES
000051  201
000052                  STAX    D               ; STORE LS BYTE SUM IN MEMORY
000052  022
000053                  INX     H               ; INCREMENT MEMORY POINTERS
000053  043
000054                  INX     D
000054  023
000055                  MOV     A,M             ; GET MS BYTE FROM MEMORY
000055  176
000056                  ADC     B               ; ADD MS BYTES
000056  210
000057                  STAX    D               ; STORE SUM OF MS BYTES IN MEMORY
000057  022
000060                  DCX     H               ; RESTORE HL TO ORIGINAL VALUE
000060  053
000061                  DCX     D               ; RESTORE DE TO ORIGINAL VALUE
000061  033
000062                  RET                     ; RETURN
000062  311
```

Program A(3) Double Precision Add With Memory Subroutine.

```
                        DOUBLE PRECISION SUBTRACT WITH MEMORY SUBROUTINE
                ;                                                                       )
                ;   (BC)   -   (M<HL>)   ---->   (M<DE>)   CY=1 INDICATES UNDERFLOW    )
                ;   16 BITS - 16 BITS = 16 BITS  -  ACCUMULATOR AND FLAGS               )
                ;   DESTROYED   -   93 STATES, 11 BYTES ROM, 2 BYTES STACK.             )
                ;                                                                       )
000063          DSUBM:
000063                  MOV     A,C             ; GET LS BYTE
000063  171
000064                  SUB     M               ; SUBTRACT LS BYTES
000064  226
000065                  STAX    D               ; STORE RESULT IN MEMORY ADDRESSED BY DE
000065  022
000066                  INX     H               ; INCREMENT MEMORY POINTER
000066  043
000067                  INX     D               ; INCREMENT DE POINTER
000067  023
000070                  MOV     A,B             ; GET NEXT BYTE
000070  170
000071                  SBB     M               ; SUBTRACT MEMORY
000071  236
000072                  STAX    D               ; STORE RESULT IN MEMORY ADDRESSED BY DE
000072  022
000073                  DCX     H               ; RESTORE HL TO ORIGINAL VALUE
000073  053
000074                  DCX     D               ; RESTORE DE TO ORIGINAL VALUE
000074  033
000075                  RET                     ; RETURN
000075  311
```

Program A(4) Double Precision Subtract With Memory Subroutine.

```
                        ;                DOUBLE PRECISION COMPARE SUBROUTINE                ;
                        ;                                                                   ;
                        ;    (M<HL>) COMPARED TO (BC)    CY=1 IMPLIES (M<HL>) < (BC)        ;
                        ;                                Z=1  IMPLIES (M<HL>) = (BC)        ;
                        ;                                                                   ;
                        ;    ACCUMULATOR AND FLAGS DESTROYED  --  62-73 STATES, 12          ;
                        ;    BYTES ROM, AND 4 BYTES OF STACK USAGE.                         ;
                        ;                                                                   ;
                        DCMPM:
                                PUSH    D               ; SAVE DE
000433      325
                                INX     H               ; POINT HL TO MS BYTE OF MEMORY TO BE COMPARED
000434
000434      043
                                MOV     A,M             ; GET MOST SIG BYTE OF MEMORY INTO ACC
000435
000435      176
                                DCX     H               ; POINT HL BACK AT LS BYTE
000436
000436      053
                                CMP     B               ; COMPARE MOST SIG BYTE OF MEM TO B
000437
000437      270
                                JNZ     EXPB            ; MEMORY NOT EQUAL TO BC SO EXIT
000440
000440      302     045     001
                                MOV     A,M             ; ELSE TEST LS BYTE
000443
000443      176
                                CMP     C               ; COMPARE LS BYTES
000444
000444      271
                        EXPB:   POP     D               ; RESTORE DE
000445
000445      321
                                RET                     ; RETURN
000446
000446      311
```

Program A(5) Double Precision Compare Subroutine.

```
                        ;                DOUBLE PRECISION MULTIPLY SUBROUTINE                 ;
                        ;                                                                     ;
                        ; THIS ROUTINE EXPECTS TWO 16-BIT NUMBERS IN THE REGISTER PAIRS      ;
                        ;  DE AND BC.  THE PRODUCT IS RETURNED AS A 4-BYTE, 32-BIT NUM-      ;
                        ;   IN MEMORY AT THE STARTING ADDRESS PRESET IN THE HL REGISTER      ;
                        ;    PAIR.     ALL CPU STATUS IS PRESERVED.  THE PRODUCT IS RE-      ;
                        ;     RETURNED LEAST SIGNIFICANT BYTE AT THE PRESET HL ADDRESS      ;
                        ;      AND UPWARDS IN MEMORY TO THE MOST SIGNIFICANT BYTE.          ;
                        ;                                                                     ;
                        ;   73 BYTES ROM, 14 BYTES STACK AND 4683-4843 MACHINE STATES.      ;
                        ;                                                                     ;
000076                  DPMULT:
000076                          PUSH    PSW             ; SAVE ALL CPU STATUS
000076  365
000077                          PUSH    B
000077  305
000100                          PUSH    D
000100  325
000101                          PUSH    H
000101  345
000102                          PUSH    B               ; SAVE FIRST FACTOR ON THE STACK
000102  305
000103                          MVI     B,16.           ; PRESET BIT SHIFT COUNT TO 16
000103  006     020
000105                          LXI     H,0             ; CLEAR HL
000105  041     000     000
000110                          SHLD    SPAD+32         ; CLEAR UPPER BYTES OF PARTIAL PRODUCT
000110  042     032     052
000113                  DPM1:   POP     H               ; 1ST FACTOR ----> HL
000113  341
000114                          CALL    DPSHRC          ; SHIFT FIRST FACTOR RIGHT THRU CARRY
000114  315     200     000
000117                          PUSH    H               ; SAVE SHIFTED FACTOR IN STACK
000117  345
000120                          LHLD    SPAD+32         ; GET TWO TOP BYTES OF PRODUCT
000120  052     032     052
000123                          JNC     DPM2            ; PRESENT LSB OF 1ST FACTOR NOT A "1"
000123  322     127     000
000126                          DAD     D               ; PRESENT LSB OF 1ST FACTOR IS A ONE SO ADD SECOND FACTOR
000126  031
000127                  DPM2:   CALL    DPSHRC          ; SHIFT UPPER PARTIAL PRODUCT RIGHT THRU CARRY
000127  315     200     000
000132                          SHLD    SPAD+32         ; SAVE IT IN MEMORY
000132  042     032     052
000135                          LHLD    SPAD+30         ; GET LOWER BYTES OF PARTIAL PRODUCT
000135  052     030     052
000140                          CALL    DPSHRC          ; SHIFT LOWER BYTES
000140  315     200     000
000143                          SHLD    SPAD+30         ; SAVE IN MEMORY
000143  042     030     052
000146                          DCR     B               ; DECREMENT SHIFT COUNT
000146  005
000147                          JNZ     DPM1            ; NOT 16TH SHIFT, GO AGAIN
000147  302     113     000
000152                          POP     H               ; RESET STACK
000152  341
000153                          POP     D               ; GET ADDRESS TO RETURN PRODUCT IN
```

Program A(6) Double Precision Multiply Subroutine.

```
000153  321
000154                          PUSH    D               ; SAVE IN STACK
000154  325
000155                          LHLD    SPAD+30         ; LOWER PRODUCT BYTES ----> HL
000155  052     030     052
000160                          XCHG                    ; HL <----> DE
000160  353
000161                          MOV     M,E             ; MOVE BYTES TO MEMORY
000161  163
000162                          INX     H
000162  043
000163                          MOV     M,D
000163  162
000164                          XCHG                    ; HL <----> DE
000164  353
000165                          LHLD    SPAD+32         ; GET UPPER BYTES OF PRODUCT
000165  052     032     052
000170                          XCHG
000170  353
000171                          INX     H
000171  043
000172                          MOV     M,E             ; PUT UPPER BYTES IN MEMORY
000172  163
000173                          INX     H
000173  043
000174                          MOV     M,D
000174  162
000175                          JMP     EXIT            ; ALL DONE, RESTORE STATUS AND RETURN
000175  303     104     002

000200                  DPSHRC: MOV     A,H             ; DOUBLE PRECISION SHIFT RIGHT
000200  174                                             ; OF HL THROUGH THE CARRY SUBROUTINE
000201                          RAR
000201  037
000202                          MOV     H,A
000202  147
000203                          MOV     A,L
000203  175
000204                          RAR
000204  037
000205                          MOV     L,A
000205  157
000206                          RET
000206  311
```

Program A(6) (continued)

```
;                    DOUBLE PRECISON DIVIDE SUBROUTINE                    ;
;                                                                         ;
;    THIS ROUNTINE EXPECTS A 32-BIT NUMBER IN HLDE AND DIVIDES            ;
;      IT BY THE 16-BIT NUMBER IN BC.   THE QUOTIENT IS RETURNED          ;
;      IN DE WHILE THE REMAINDER IS RETURNED IN HL.  THE ORIGINAL         ;
;        DIVISOR IS SAVED AND RETURNED IN BC.   THE QUOTIENT IS           ;
;            ROUNDED-OFF.  CY=1 IMPLIES ROUND-OFF OCCURED.                ;
;                                                                         ;
; 64 BYTES ROM + 7 BYTES FROM DPMULT ROUTINE, 8 BYTES STACK USAGE         ;
;                      AND 3645-8456 MACHINE STATES.                      ;
;                                                                         ;
000207                          DPDIV:
000207                                  MVI     A,17.           ; PRESET BIT SHIFT COUNT
000207  076     021
000211                          DPDIV1: PUSH    PSW             ; SAVE BITS SHIFT COUNT
000211  365
000212                                  CALL    DPSUB           ; DOUBLE PRECISON SUBTRACT (HL-BC)
000212  315     257     000
000215                                  XCHG                    ; HL <----> DE
000215  353
000216                                  CALL    DPSHLC          ; SHIFT HL LEFT THRU CARRY
000216  315     300     000
000221                                  XCHG                    ; HL <----> DE
000221  353
000222                                  CALL    DPSHLC          ; SHIFT HL LEFT THRU CARRY
000222  315     300     000
000225                                  POP     PSW             ; GET BITS SHIFT COUNT
000225  361
000226                                  DCR     A               ; DECREMENT SHIFT COUNT
000226  075
000227                                  JNZ     DPDIV1          ; NOT 16TH SHIFT, LOOP
000227  302     211     000
000232                                  PUSH    B               ; SAVE DIVISOR
000232  305
000233                                  PUSH    H               ; SAVE REMAINDER
000233  345
000234                                  MOV     L,C             ; DIVISOR TO HL
000234  151
000235                                  MOV     H,B
000235  140
000236                                  CALL    DPSHRC          ; DIVIDE DIVISOR BY TWO
000236  315     200     000
000241                                  MOV     B,H             ; DIVISOR/2 ----> BC
000241  104
000242                                  MOV     C,L
000242  115
000243                                  POP     H               ; RESTORE REMAINDER
000243  341
000244                                  PUSH    H               ; SAVE REMAINDER AGAIN
000244  345
000245                                  CALL    DPSUB           ; REMAINDER-(DIVISOR/2) ----> HL
000245  315     257     000
000250                                  JNC     DPDIV2          ; FIRST FRACTION < .5 THEREFORE DON'T ROUNDOFF
000250  322     254     000
000253                                  INX     D               ; ELSE ROUND-OFF QUOTIENT
000253  023
```

Program A(7) Double Precision Divide Subroutine.

```
000254                          DPD1V2: POP     H               ; RESTORE REMAINDER
000254  341
000255                                  POP     B               ; RESTORE DIVISOR
000255  301
000256                                  RET                     ; RETURN TO CALLING POINT
000256  311

000257                          DPSUB:  MOV     A,L             ; FIRST TEST BC>HL?
000257  175
000260                                  SUB     C
000260  221
000261                                  MOV     A,H
000261  174
000262                                  SBB     B
000262  230
000263                                  JNC     DPSUB1
000263  322     270     000
000266                                  CMC
000266  077
000267                                  RET
000267  311
000270                          DPSUB1: MOV     A,L             ; IF HERE, BC<=HL SO SUBTRACT
000270  175
000271                                  SUB     C
000271  221
000272                                  MOV     L,A
000272  157
000273                                  MOV     A,H
000273  174
000274                                  SBB     B
000274  230
000275                                  MOV     H,A             ; DIFFERENCE RETURNED TO HL
000275  147
000276                                  STC                     ; INDICATE SUBTRACTION MADE BY SETTING CARRY
000276  067
000277                                  RET
000277  311

000300                          DPSHLC: MOV     A,L             ; DOUBLE PRECISION SHIFT LEFT OF
000300  175
000301                                  RAL                     ; HL THROUGH THE CARRY SUBROUTINE
000301  027
000302                                  MOV     L,A
000302  157
000303                                  MOV     A,H
000303  174
000304                                  RAL
000304  027
000305                                  MOV     H,A
000305  147
000306                                  RET
000306  311
```

Program A(7) (continued)

```
                                ;                    SQUARE ROOT SUBROUTINE                    ;
                                ;                                                              ;
                                ;    THIS ROUTINE EXPECTS A MAXIMUM 16 BIT QUANTITY IN HL AND  ;
                                ;  RETURNS THE SQUARE ROOT IN THE ACCUMULATOR.  ALL CPU STATUS ;
                                ;    IS PRESERVED EXCEPT ACCUMULATOR AND FLAGS.  THE ROOT IS   ;
                                ;     ROUNDED-OFF TO PROVIDE A MORE ACCURATE INTEGER RESULT.   ;
                                ;                                                              ;
                                ; 84 BYTES ROM + 24 BYTES FROM DPDIV ROUTINE, 14 BYTES OF STACK ;
                                ;                 USEAGE AND 4336-4746 MACHINE STATES.         ;
000307                          SQRT:
000307                                  PUSH    B               ; SAVE CPU STATUS
000307  305
000310                                  PUSH    D
000310  325
000311                                  PUSH    H
000311  345
000312                                  LXI     B,8.            ; 0 ----> B  8. ----> C
000312  001     010     000
000315                                  LXI     D,0             ; CLEAR DE
000315  021     000     000
000320                          SQRT1:  PUSH    B               ; SAVE SHIFT COUNT AND PARTIAL ROOT
000320  305
000321                                  PUSH    H               ; SAVE NUMBER
000321  345
000322                                  XRA     A               ; CLEAR A AND CARRY
000322  257
000323                                  MOV     L,B             ; PARTIAL ROOT TO L
000323  150
000324                                  MOV     H,A             ; 0 ----> H
000324  147
000325                                  CALL    DPSHLC          ; SHIFT HL LEFT INTO CARRY
000325  315     300     000
000330                                  STC                     ; SET CARRY
000330  067
000331                                  CALL    DPSHLC          ; SHIFT 1 INTO HL
000331  315     300     000
000334                                  MOV     B,H             ; PUT HL INTO BC
000334  104
000335                                  MOV     C,L
000335  115
000336                                  POP     H               ; RESTORE NUMBER
000336  341
000337                                  CALL    DPSHLC          ; SHIFT TWO BITS FROM HL INTO DE PARTIAL DIFFERENCE
000337  315     300     000
000342                                  XCHG
000342  353
000343                                  CALL    DPSHLC          ; SHIFT HL LEFT
000343  315     300     000
000346                                  XCHG
000346  353
```

Program A(8) Square Root Subroutine.

```
000047                          CALL    DPSHLC          ; SHIFT HL LEFT
000047  315     300     000
000052                          XCHG
000052
000053                          CALL    DPSHLC          ; SHIFT HL LEFT
000053  315     300     000
000056                          CALL    DPSUB           ; SUBTRACT PARTIAL ROOT WITH 01
000056  315     257     000
                                                        ; ADDED FROM PARTIAL DIFFERENCE LEFT IN HL
000061                          XCHG                    ; PUT PARTIAL DIFFERENCE BACK IN DE
000061  350
000062                          POP     B               ; GET PARTIAL ROOT BACK AND SHIFT COUNT
000062  301
000063                          MOV     A,B             ; SHIFT IN NEXT BIT OF ROOT
000063  170
000064                          RAL
000064  027
000065                          MOV     B,A             ; SAVE PARTIAL ROOT
000065  107
000066                          DCR     C               ; DECREMENT SHIFT COUNT
000066  015
000067                          JNZ     SQRT1           ; NOT DONE, DO AGAIN
000067  302     320     000
000072                          XRA     A               ; CLEAR CARRY AND ACC AND CHECK FOR ROUND-OFF
000072  257
000073                          PUSH    B               ; SAVE PARTIAL ROOT
000073  305
000074                          MOV     L,B             ; PARTIAL ROOT---->L
000074  150
000075                          MOV     H,A             ; 0---->H
000075  147
000076                          CALL    DPSHLC          ; SHIFT 0 INTO PARTIAL ROOT
000076  315     300     000
000401                          STC                     ; SET CARRY
000401  067
000402                          CALL    DPSHLC          ; SHIFT 1 INTO PARTIAL ROOT TO CREATE R01
000402  315     300     000
000405                          MOV     B,H             ; PUT R01 INTO BC
000405  104
000406                          MOV     C,L
000406  115
000407                          XCHG                    ; GET PARTIAL DIFFERENCE FROM DE
000407  353
000410                          CALL    DPSHLC          ; SHIFT 00 INTO HL
000410  315     300     000
000413                          CALL    DPSHLC
000413  315     300     000
000416                          CALL    DPSUB           ; SUBTRACT HL-BC IF POSSIBLE
000416  315     257     000
000421                          POP     B               ; RESTORE PARTIAL ROOT
000421  301
000422                          JNC     SQRT2           ; NO CARRY IMPLIES NO ROUNDOFF REQUIRED
000422  322     026     001
000425                          INR     B               ; ELSE ROUND OFF UPWARD
000425  004
```

Program A(8) (continued)

```
000426                  SQRT2:  MOV     A,B             ; FINISHED, PUT ROOT IN A
000426  170
000427                          POP     PSW             ; RETORE STATUS
000427  341
000430                          POP     D
000430  321
000431                          POP     B
000431  301
000432                          RET                     ; RETURN
000432  311
```

Program A(8) (continued)

```
;       MULTIBYTE BINARY ADDITION AND SUBTRACTION SUBROUTINES
;
;  SUBROUTINES MADD AND MSUB PERFORM MULTIBYTE, UNSIGNED BINARY
;   ADDITION AND SUBTRACTION ON TWO NUMBERS A AND B.    THE TWO
;    NUMBERS TO BE ADDED (OR SUNTRACTED) ARE STORED LOW-ORDER
;     BYTE TO HIGH-ORDER BYTE BEGINNING AT <BC> AND <HL> RES-
;      PECTIVELY.  NUMBER B IS THE SUBTRAHEND FOR SUBTRACTION.
;        THE RESULT IS STORED LOW-ORDER BYTE TO HIGH-ORDER
;         BYTE IN THE LOCATION FORMERLY OCCUPIED BY THE
;          NUMBER A.   REGISTER E SPECIFIES THE NUMBER
;           OF BYTES IN EACH NUMBER.   EXECUTION TIME
;            'T' IS GIVEN BY:  T=(E)56 + 25 STATES
;             WHERE E IS THE # BYTES PER NUMBER
```

```
000000                          MADD:                   ; SUBROUTINE - MULTIBYTE ADD
000000                                  PUSH    PSW     ; SAVE CPU STATUS
000000  365
000001                                  PUSH    B
000001  305
000002                                  PUSH    D
000002  325
000003                                  PUSH    H
000003  345
000004                                  XRA     A       ; CLEAR ACCUMULATOR AND CARRY BIT
000004  257
000005                                  LDAX    B       ; LOAD BYTE OF FIRST NUMBER TO A
000005  012
000006                                  ADC     M       ; ADD WITH CARRY BYTE OF SECOND NUMBER
000006  216
000007                                  STAX    B       ; STORE SUM IN [(BC)]
000007  002
000010                                  DCR     E       ; ADDITION FINISHED IF E=0
000010  035
000011                                  JZ      EXIT    ; JUMP IF FINISHED
000011  312     147     000
000014                                  INX     B       ; OTHERWISE POINT BC TO NEXT BYTE OF 1ST NUMBER
000014  003
000015                                  INX     H       ; ALSO POINT HL TO NEXT BYTE OF 2ND NUMBER
000015  043
000016                                  JMP     MADD+5  ; ADD NEXT TWO BYTES
000016  303     005     000

000021                          MSUB:                   ; START OF MULTIBYTE SUBTRACT ROUTINE
000021                                  PUSH    PSW     ; SAVE CPU STATUS
000021  365
000022                                  PUSH    B
000022  305
000023                                  PUSH    D
000023  325
000024                                  PUSH    H
000024  345
000025                                  XRA     A       ; CLEAR ACCUMULATOR AND CARRY BIT
000025  257
```

Program A(9) Mulitbyte Binary Addition and Subtraction Subroutines.

```
000026                          LDAX    B               ; LOAD BYTE OF MINUEND TO A
000026  012
000027                          SBB     M               ; SUBTRACT BYTE OF SUBTRAHEND FROM A
000027  236
000030                          STAX    B               ; STORE DIFFERENCE IN [(BC)]
000030  002
000031                          DCR     E               ; ALL DONE IF E=0
000031  035
000032                          JZ      EXIT            ; JUMP IF FINISHED
000032  312     147     000
000035                          INX     B               ; OTHERWISE POINT (BC) TO NEXT BYTE
000035  003
000036                          INX     H               ; AND POINT (HL) TO NEXT BYTE
000036  043
000037                          JMP     MSUB+5          ; SUBTRACT NEXT TWO BYTES
000037  303     026     000
```

Program A(9) (continued)

```
;                    MULTIBYTE COMPARE SUBROUTINE                    ;
;                                                                    ;
; SUBROUTINE MCMP COMPARES TWO MULTIBYTE UNSIGNED BINARY NUMBERS     ;
;   A AND B WHICH ARE STORED LOW-ORDER BYTE TO HIGH-ORDER BYTE       ;
;     BEGINNING AT <BC> AND <HL> RESPECTIVELY.  REGISTER E           ;
;       SPECIFIES THE NUMBER OF BYTES IN EACH NUMBER.  THE           ;
;        RESULT OF THE COMPARISON IS RETURNED IN REGISTER            ;
;          D.  FOR A>B, D=2.  FOR A<B, D=1.  FOR A=B, D=0.          ;
;            TIMING HAS BEEN ADJUSTED SO THAT EXECUTION             ;
;             TIME DEPENDS ONLY ON THE NUMBER OF BYTES              ;
;              IN EACH NUMBER.  EXECUTION TIME T IS:               ;
;               T = 110*E + 108 STATES  WHERE E =                  ;
;                    NUMBER OF BYTES PER NUMBER                    ;

000042                          MCMP:                               ; START OF MULTIBYTE COMPARE ROUTINE
000042                                  PUSH    PSW                 ; SAVE MACHINE STATUS
000042  365
000043                                  PUSH    B
000043  305
000044                                  PUSH    D
000044  325
000045                                  PUSH    H
000045  345
000046                                  MVI     D,0                 ; CLEAR FLAG BYTE
000046  026     000
000050                                  LDAX    B                   ; LOAD BYTE OF NUMBER A
000050  012
000051                                  CMP     M                   ; COMPARE TO BYTE OF NUMBER B
000051  276
000052                                  JC      ALTM                ; JUMP IF BYTE A < BYTE B
000052  332     073     000
000055                                  JNZ     AGTM                ; JUMP IF BYTE  A > BYTE B
000055  302     100     000
000060                                  MVI     A,0                 ; BYTE A = BYTE B
000060  076     000
000062                                  CMP     D                   ; IF D=0 THIS MUST BE THE FIRST BYTE COMPARISON
000062  272
000063                                  JNZ     AGTM+5              ; IF NOT EQUAL, DO NOT CHANGE FLAG BYTE E
000063  302     105     000
000066                                  MVI     D,3                 ; IF THIS IS THE FIRST BYTE COMPARISON AND
000066  026     003
                                                                    ; A=B, SET THE FLAG
000070                                  JMP     AGTM+3.
000070  303     110     000
```

Program A(10) Multibyte Compare Subroutine.

```
000073                          ALTM:
000073                                  MVI     D,1             ; ACD SO SET D=1
000073  026     001
000075                                  JMP     AGTM+2
000075  303     102     000
000100                          AGTM:
000100                                  MVI     D,2             ; A>B SO SET D=2
000100  026     002
000102                                  MOV     A,M             ; DELAY 7 STATES. NEXT THREE INSTRUCTIONS
000102  176
000103                                  DLY10                   ; ALSO PROVIDE DELAY TO EQUALIZE EXECUTION TIMES
000103  177     177
000105                                  MOV     A,M             ; DELAY 7 MORE STATES
000105  176
000106                                  DLY10
000106  177     177
000110                                  DCR     E               ; COMPARISON COMPLETE IF E=0
000110  035
000111                                  JZ      MC1             ; JUMP IF THROUGH
000111  312     121     000
000114                                  INX     H               ; OTHERWISE POINT (HL) TO NEXT BYTE OF NUMBER A
000114  043
000115                                  INX     B               ; POINT (BC) TO NEXT BYTE OF NUMBER B
000115  003
000116                                  JMP     MCMP+6          ; COMPARE NEXT TWO BYTES
000116  303     050     000
000121                          MC1:
000121                                  POP     H               ; RESTORE MACHINE STATE EXCEPT FOR
000121  341
000122                                  MOV     B,D             ; FLAG REGISTER E
000122  102
000123                                  POP     D
000123  321
000124                                  MOV     D,B
000124  120
000125                                  POP     B
000125  301
000126                                  POP     PSW
000126  361
000127                                  RET                     ; RETURN TO CALLING POINT
000127  311
```

Program A(10) (continued)

```
;                    MULTIBYTE MEMORY-TO-MEMORY MOVE                    ;
; SUBROUTINE MOVE DUPLICATES A BLOCK OF MEMORY BEGINNING AT <BC>  ;
;   IN A BLOCK OF MEMORY BEGINNING AT <HL>. REGISTER E SPECIFIES  ;
;    THE NUMBER OF BYTES IN THE BLOCK.  EXECUTION TIME 'T' IS :   ;
;     T = 49*E + 91 STATES WHERE E = NUMBER OF BYTES IN BLOCK.    ;

000130                          MOVE:                   ; STARTING POINT OF MOVE BLOCK ROUTINE
000130                                  PUSH    PSW     ; SAVE CPU STATUS
000130  365
000131                                  PUSH    B
000131  305
000132                                  PUSH    D
000132  325
000133                                  PUSH    H
000133  345
000134                                  LDAX    B       ; LOAD BYTE OF DATA BLOCK
000134  012
000135                                  MOV     M,A     ; MOVE TO NEW MEMORY LOCATION
000135  167
000136                                  DCR     E       ; MOVE COMPLETE IF E=0
000136  035
000137                                  JZ      EXIT    ; RETURN IF THROUGH
000137  312     147     000
000142                                  INX     B       ; OTHERWISE POINT (BC) TO NEXT BYTE
000142  003
000143                                  INX     H       ; POINT (HL) TO NEXT BYTE OF NEW BLOCK
000143  043
000144                                  JMP     MOVE+4  ; MOVE THE NEXT BYTE
000144  303     134     000
```

Program A(11) Multibyte Memory-to-Memory Move Subroutine.

```
                        ;
                        ;                   GENERAL PURPOSE EXIT ROUTINE
                        ;
                        ; THIS ROUTINE POPS ALL CPU STATUS OFF THE STACK AND THUS IT
                        ; RESTORES ALL MACHINE STATUS ASSUMING STATUS WAS ORIGINALLY
                        ; PUSHED ONTO THE STACK.  60 MACHINE STATES INCLUDING THE JUMP
                        ; INSTRUCTION USED TO ACCESS THIS ROUTINE ARE REQUIRED FOR ITS
                        ;                           EXECUTION.
                        ;
001104                  EXIT:                       ; GENERAL PURPOSE EXIT ROUTINE
001104                          POP     H           ; RESTORES MACHINE STATUS
001104  341
001105                          POP     D
001105  321
001106                          POP     B
001106  301
001107                          POP     PSW
001107  361
001110                          RET                 ; RETURN
001110  311

001111                          END
```

Program A(12) General Purpose Exit Subroutine.

APPENDIX B

A NEW PRINCIPLE FOR FAST FOURIER TRANSFORMATION

APPENDIX B

A NEW PRINCIPLE FOR FAST FOURIER TRANSFORMATION

by

C. M. Rader, N. M. Brenner

Let $\{a_n\}$ be a sequence of $N = 2^m$ data, whose Discrete Fourier Transform is $\{A_k\}$. Let $W = \exp(-j2\pi/N)$. Present FFT algorithms are derived from an equation like (1) or its dual*

$$A_k = \mathrm{DFT}\{a_{2n}\} + W^k * \mathrm{DFT}\{a_{2n+1}\} . \qquad (1)$$

Each of the DFT's in (1) is a DFT of a half-length data sequence, and can be expressed as two still shorter DFT's. After m such stages of simplification an algorithm is evident which requires $5(N \log_2 N)$ operations to execute.

This note presents an alternative to Equation (1) which may similarly be applied iteratively to itself leading to an FFT algorithm. The new algorithm which so results has the peculiarity that none of the multiplying constants is complex. Its advantages would therefore seem to be most pronounced in systems for which multiplications are most costly.

---

*Equation (1) is too specialized, leading to radix-2 algorithms. It portrays the essence of the derivation, however.

Derivation

Let $\{b_n\}$ and $\{c_n\}$ be the sequences

$$b_n = a_{2n}$$
$$c_n = a_{2n+1} - a_{2n-1} + Q \qquad n = 0, 1, \ldots, N/2 - 1 \qquad (2)$$

where

$$Q = \frac{2}{N} \sum_{n=0}^{\frac{N}{2}-1} a_{2n+1}$$

The $\frac{N}{2}$ points DFT's of these sequences are $\{B_k\}$ and $\{C_k\}$. It is helpful to consider the sequence $\{d_n\}$ and its DFT, $\{D_k\}$ defined by

$$d_n = a_{2n+1} \qquad n = 0, 1, \ldots, N/2 - 1 \qquad (3)$$

so that $\{B_k\}$ and $\{D_k\}$ are the DFT's in Equation (1).

C_k and D_k can be simply related. Since Q is a constant, it appears in C_k in only one term, C_o. Furthermore, C_o is exactly equal to D_o. For other values of k, C_k can be expressed by the circular shifting theorem for DFT.

$$C_k = D_k(1 - W^{2k})$$
$$= W^{-k} D_k (W^k - W^{-k})$$
$$k = 1, 2, \ldots, \frac{N}{2} - 1 \qquad (4)$$

Hence,

$$W^k D_k = C_k/(W^k - W^{-k}) \qquad k \neq 0 \qquad (5)$$

and we can rewrite Equation (1)

$$A_k = B_k + C_k/(W^k - W^{-k})$$

$$k = 1, 2, \ldots, \frac{N}{2} - 1, \frac{N}{2} + 1, \frac{N}{2} + 2, \ldots, N - 1 \qquad (6a)$$

$$\left.\begin{aligned} A_o &= B_o + C_o \\ A_{N/2} &= B_{N/2} - C_{N/2} \end{aligned}\right\} \qquad (6b)$$

Since W^k and W^{-k} are complex conjugates, their difference is twice the imaginary part, e.g.,

$$W^k - W^{-k} = -2j \sin 2\pi k/N$$

so that (6a) may be written

$$A_k = B_k + \frac{1}{2}j \csc(\frac{2\pi}{N} k) \quad C_k \qquad k \neq 0, N/2 \qquad (6a)$$

Equations (6a) and (6b) are the replacements for Equation (1) promised. A fast Fourier Transform algorithm based on (6a) and (6b) requires only multiplication by pure imaginary constants.

Comments

$\frac{N}{2}Q$ could have been added to the output terms A_o, $A_{N/2}$ rather than to each of $\frac{N}{2}$ input terms as in Equation (2) but this would have made the substitution of Equation (2) recursively into itself, to produce an FFT algorithm, a difficult matter. In hardware implementation, e.g., "pipelines" this may not be a consideration.

If, in Equation (2) we had used $a_{2n+1} + a_{2n-1}$, the minus sign would change to + in Equations (4), (5), and (6a), thus changing $\frac{1}{2}j \csc(2\pi k/N)$ to $\frac{1}{2} \sec(2\pi k/N)$. The exceptional cases for k would then be k = N/4, 3N/4 and these would involve $\pm j$, the only non-real operation encountered. We judge that the secant form offers no advantages over the cosecant form.

Whereas the constants W^k used in Equation (1) have unity magnitude, $\csc(2\pi k/N)$ can get very large. Therefore, small computation errors can lead to large output errors. Experience verifies that this is the case. We recommend that the method be modified if more than 8192 points are used. A conventional factoring can reduce a DFT to shorter DFT's and each of these can be computed by the cosecant method.

Substantial savings in multiplications can be made in the conventional FFT by deriving the algorithm in a higher radix. The method proposed here can also be developed for radices other than two. The (p-1) sequences to be formed are of the form $c_n^{(q)} = a_{pn+q} - a_{pn-q} + Q_q$. We have no idea whether computational savings can result from higher radix methods.

APPENDIX C

8080A FAST FOURIER TRANSFORM PROGRAM

```
                        ;PDCP 8080 FFT PROGRAM
;
;   THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON AN ARRAY OF
;   REAL DATA STORED IN THE COMPLEX ARRAY 'DATA'.  AS WRITTEN,
;   IT CAN PERFORM UP TO 64-POINT TRANSFORMS MERELY BY CHANGING N3,
;   WHICH IS 2*(NUMBER OF POINTS OF THE TRANSFORM) AND LOG2N, WHERE
;   2**LOG2N=N3.  THE ALGORITHM USED WAS DEVELOPED BY C. RADER AND
;   N. BRENNER, FORMERLY OF MIT, AND HAS BEEN ADAPTED TO RUN
;   ON AN 8080 BASED MACHINE.  THIS ALGORITHM IS IDEALLY SUITED
;   TO THE 8080 SINCE IT REQUIRES FEWER MULTIPLICATIONS THAN THE
;   STANDARD FFT FORM.  REGISTERS ARE USED TO HOLD VARIABLES WHEREVER
;   POSSIBLE TO AVOID THE TIME CONSUMED BY MEMORY ACCESS.  ALSO,
;   A SPECIAL MULTIPLICATION ROUTINE HAS BEEN IMPLEMENTED,
;   RESULTING IN A VERY FAST TRANSFORM ROUTINE.  ALL INSTRUCTIONS
;   WHICH MUST BE ALTERED WHEN CHANGING THE NUMBER OF POINTS IN
;   THE TRANSFORM ARE SET OFF BY ***************.
;
;   THE MACRO CPXTBL FINDS THE (R/2)'TH VALUE (2 BYTES) OF TBLE.
;   THE VALUE IS STORED IN VAL, AND THE ADDRESS OF THE VALUE IN ADDR.
;   HAVING THE VALUE ADDRESS IN ADDR PERMITS A FORM OF INDIRECT
;   ADDRESSING.

.MACRO  CPXTBL  TBLE,R,ADDR,VAL
        .NLIST  SRC
        LXI     H,TBLE  ;HL <- TBLE
        MOV     A,L     ;L IS LSB OF TBLE
        ADD     R       ;A <- LSB(TBLE)+(R)
        MOV     L,A
        JNC     $+^04>
        INR     H       ;INCREMENT MSB IF CARRY
        SHLD    ADDR    ;ADDR <- TBLE+(R)
        MOV     A,M     ;A <- REAL(TBLE(R))
        INX     H
        MOV     L,M     ;L <- IMAG(TBLE(R))
        MOV     H,A     ;HL <- (TBLE(R))
        SHLD    VAL     ;VAL <- (TBLE(R))
        .LIST   SRC
.ENDM
;
;
;
;  ******************************
000037  N3      =       31.             ;N3=2*NUMBER OF POINTS IN TRANSFORM + 1
000004  LOG2N   =       4               ;2**LOG2N=NUMBER OF POINTS OF TRANSFORM
;  ******************************
;  ADD UP STAGE
;  REGISTER B HOLDS THE VARIABLE I
;  REGISTER C HOLDS THE VARIABLE J
;
;  REGISTER D HOLDS THE VARIABLE N1
;  REGISTER E HOLDS THE VARIABLE N2
;
      ;N1=1
000000  FFT80:  MVI     D,2             ;2 BYTES/ CMPLEX.  VALUE, D=N1
000000  026  002
      ;K=LOG2N
000002    LXI     H,K
```

Program C(1) 8080A Fast Fourier Transform Program (FFT80).

```
000002  041  351  004
000005                        MVI     M,LOG2N
000005  066  004
                        ;10   K=K-1
000007                  S10:  LXI     H,K
000007  041  351  004
000012                        DCR     M
000012  065
                              ;IF(K.EQ.0) GO TO 40
000013                        JZ      S40
000013  312  017  001
                        ;12   N2=N1+N1
000016                  S12:  MOV     A,D               ;PUT N1 IN A
000016  172
000017                        ADD     A                 ;A <- 2*N1
000017  207
000020                        MOV     E,A               ;N2 <- A, E=N2
000020  137
                              ;I=N1+1
000021                        MOV     A,D
000021  172
000022                        ADI     2
000022  306  002
000024                        MOV     B,A
000024  107
                        ;15   J=1-N2+N3+1
000025                  S15:  MOV     A,B               ;PUT I IN A, B=I
000025  170
000026                        ADI     N3                ;A <- I+N3
000026  306  037
000030                        SUB     E                 ;A <- I+N3-N2
000030  223
000031                        INR     A
000031  074
000032                        MOV     C,A               ;STORE IN J, C=J
000032  117
                              ;PREV=DATA(J)
000033                        CPXTBL  DATA,C,DJADDR,DJVAL       ;HL <- (DATA(J))
000033  041  040  005
000036  175
000037  201
000040  157
000041  322  045  000
000044  044
000045  042  026  005
000050  176
000051  043
000052  156
000053  147
000054  042  034  005
000057                        SHLD    PREV
000057  042  016  005
                              ;SUM=0.0
000062                        MVI     H,0
000062  046  000
000064                        MOV     L,H               ;HL <- 0
000064  154
```

Program C(1) (continued)

```
000065                                  SHLD    SUM
000065  042     020     005
000070                                  SHLD    SUM+2           ;(SUM) . . . (SUM+3) <- 0
000070  042     022     005
                                ;J=I
000073                                  MOV     C,B
000073  110
                        ;20     SUM=SUM+DATA(J)
000074                          S20:    CPXTBL  DATA,C,DJADDR,DJVAL     ;HL <- (DATA(J))
000074  041     040     005
000077  175
000100  201
000101  157
000102  022     106     000
000105  044
000106  042     026     005
000111  176
000112  043
000113  156
000114  147
000115  042     034     005
000120                                  XCHG                    ;DE <- (DATA(J))
000120  353
000121                                  PUSH    H               ;SAVE N1 & N2 ON STACK
000121  345
000122                                  PUSH    B               ;SAVE I & J
000122  305
000123                                  LHLD    SUM             ;HL <- REAL(SUM)
000123  052     020     005
000126                                  MOV     C,D
000126  112
000127                                  MVI     B,0             ;BC <- REAL(DATA(J))
000127  006     000
000131                                  DAD     B
000131  011
000132                          OVRR:   SHLD    SUM
000132  042     020     005
000135                                  LHLD    SUM+2           ;HL <- IMAG(SUM)
000135  052     022     005
000140                                  MOV     C,E
000140  113
000141                                  DAD     B
000141  011
000142                          OVRI:   SHLD    SUM+2
000142  042     022     005
000145                                  POP     B
000145  301
000146                                  POP     H               ;RESTORE N1 & N2
000146  341
000147                                  XCHG                    ;HL <- (DATA(J)), N1&N2 RESTORED
000147  353
                                ;TEMP=DATA(J)
000150                                  SHLD    TEMP
000150  042     024     005
                                ;TEMP=TEMP-PREV
000153                                  MOV     A,H             ;H CONTAINS REAL(TEMP)
000153  174
```

Program C(1) (continued)

```
000154                          LXI     H,PREV+1
000154  041     017     005
000157                          SUB     M               ;A <- REAL(TEMP-PREV)
000157  226
000160                          LXI     H,TEMP+1
000160  041     025     005
000163                          MOV     M,A
000163  167
000164                          DCX     H               ;HL POINTS TO IMAG(TEMP)
000164  053
000165                          MOV     A,M             ;A <- IMAG(TEMP)
000165  176
000166                          LXI     H,PREV
000166  041     016     005
000171                          SUB     M
000171  226
000172                          LXI     H,TEMP
000172  041     024     005
000175                          MOV     M,A
000175  167

                        ;PREV=DATA(J)
000176                          LHLD    DJVAL
000176  052     034     005
000201                          SHLD    PREV
000201  042     016     005

                        ;DATA(J)=TEMP
000204                          LHLD    TEMP
000204  052     024     005
000207                          PUSH    D
000207  325
000210                          XCHG
000210  353
000211                          LHLD    DJADDR
000211  052     026     005
000214                          MOV     M,D
000214  162
000215                          INX     H
000215  043
000216                          MOV     M,E
000216  163
000217                          POP     D
000217  321

                        ;J=J+N2
000220                          MOV     A,C
000220  171
000221                          ADD     E
000221  203
000222                          MOV     C,A
000222  117

                        ;IF(J-N3-1.LE.0) GO TO 20
000223                          DCR     A
000223  075
000224                          CPI     N3
000224  376     037
000226                          JM      S20
000226  372     074     000
000231                          JZ      S20
```

Program C(1) (continued)

```
000231  312  074  000
                        ;25   SUM=SUM/2**K
000234                  S25:    PUSH    B
000234  305
000235                          PUSH    D               ;SAVE REGISTERS
000235  325
000236                          LHLD    SUM
000236  052  020  005
000241                          XCHG                    ;DE <- (SUM)(SUM+1)
000241  353
000242                          LHLD    SUM+2           ;HL <- (SUM+2)(SUM+3)
000242  052  022  005
000245                          MOV     B,H
000245  104
000246                          MOV     C,L             ;BC <- (SUM+2)(SUM+3)
000246  115
000247                          LXI     H,K
000247  041  351  004
000252                          MOV     H,M             ;PUT K IN H (LOOP COUNTER)
000252  146
000253                  LOOP25: DCR     H               ;DECREMENT LOOP COUNTER
000253  045
000254                          JM      OTLOOP          ;EXIT AFTER K LOOPS
000254  372  300  000
000257                          MOV     A,B
000257  170
000260                          ANA     A               ;RESET CARRY
000260  247
000261                          RRC                     ;ROTATE B RIGHT
000261  017
000262                          MOV     B,A
000262  107
000263                          MOV     A,C
000263  171
000264                          RAR                     ;ROTATE C THRU CARRY
000264  037
000265                          MOV     C,A
000265  117
000266                          MOV     A,D
000266  172
000267                          ANA     A
000267  247
000270                          RRC                     ;ROTATE D
000270  017
000271                          MOV     D,A
000271  127
000272                          MOV     A,E
000272  173
000273                          RAR                     ;ROTATE E THRU CARRY
000273  037
000274                          MOV     E,A
000274  137
000275                          JMP     LOOP25
000275  303  253  000
000300                  OTLOOP: XCHG                    ;HL <- REAL(SUM)/2**K
000300  353
000301                          SHLD    SUM
```

Program C(1) (continued)

```
000301  042  020  005
000304                    MOV     H,B
000304  140
000305                    MOV     L,C             ;HL <- IMAG(SUM)/2**K
000305  151
000306                    SHLD    SUM+2
000306  042  022  005
000311                    POP     D
000311  321
000312                    POP     B               ;RESTORE REGISTERS
000312  301
                        ;J=1
000313                    MOV     C,B
000313  110
                    ;30   DATA(J)=DATA(J)+SUM
000314              S30:  PUSH    D               ;SAVE N1 & N2
000314  325
000315                    LXI     H,SUM
000315  041  020  005
000320                    MOV     A,M             ;A <- REAL(SUM)
000320  176
000321                    PUSH    PSW             ;STORE A
000321  365
                    ;  AFTER DIVISION BY 2**K, SUM IS NOW A TWO BYTE COMPLEX VALUE.
000322                    CPXTBL  DATA,C,DJADDR,DJVAL     ;HL <- (DATA(J))
000322  041  040  005
000325  175
000326  201
000327  157
000330  322  334  000
000333  044
000334  042  026  005
000337  176
000340  043
000341  156
000342  147
000343  042  034  005
000346                    POP     PSW             ;RESTORE A
000346  361
000347                    ADD     H               ;A <- REAL(DATA(J)+SUM)
000347  204
000350                    MOV     D,A             ;SAVE REAL RESULT
000350  127
000351                    MOV     A,L             ;A <- IMAG(DATA(J))
000351  175
000352                    LXI     H,SUM+2
000352  041  022  005
000355                    ADD     M               ;A <- IMAG(DATA(J)+SUM)
000355  206
000356                    LHLD    DJADDR          ;HL <- DATA+(J)
000356  052  026  005
000361                    MOV     M,D
000361  162
000362                    INX     H
000362  043
000363                    MOV     M,A
000363  167
```

Program C(1) (continued)

```
000364                          POP     D               ;RESTORE N1 & N2
000364  321
                              ;J=J+N2
000365                          MOV     A,C
000365  171
000366                          ADD     E               ;A <- J+N2
000366  203
000367                          MOV     C,A
000367  117
                              ;IF(J-N3+1.LE.0) GO TO 30
000370                          DCR     A
000370  075
000371                          CPI     N3
000371  376     037
000373                          JM      S30
000373  372     314     000
000376                          JZ      S30
000376  312     314     000
                      ;35     I=I+1
000401                S35:      INR     B
000401  004
000402                          INR     B               ;2BYTES/CMPLX. VALUE
000402  004
                              ;IF(I-N2.LE.0) GO TO 15
000403                          MOV     A,B
000403  170
000404                          CMP     E
000404  273
000405                          JM      S15
000405  372     025     000
000410                          JZ      S15
000410  312     025     000
                      ;38     N1=N2
000413                S38:      MOV     D,E
000413  123
                              ;GO TO 10
000414                          JMP     S10
000414  303     007     000
                      ;
                      ;         PERMUTATION BY BIT REVERSAL
                      ;
                      ;   REGISTER B HOLDS THE VARIABLE I
                      ;   REGISTER C HOLDS THE VARIABLE J
                      ;   REGISTER D HOLDS THE VARIABLE N1
                      ;   REGISTER E HOLDS THE VARIABLE N2
                      ;40     I=1
000417                S40:      MVI     B,2
000417  006     002
                              ;J=1
000421                          MOV     C,B
000421  110
                      ;45     IF(I-J.GE.0) GO TO 50
000422                S45:      MOV     A,B
000422  170
000423                          CMP     C
000423  271
000424                          JP      S50
```

Program C(1) (continued)

```
000424  362  125  001
                        ;40   TEMP=DATA(I)
000427                  340:    CPXTBL  DATA,B,DIADDR,DIVAL      ;HL <- (DATA(I))
000427  041  040  005
000432  175
000433  200
000434  157
000435  022  041  001
000440  044
000441  042  030  005
000444  176
000445  043
000446  156
000447  147
000450  042  036  005
000453                    SHLD    TEMP
000453  042  024  005
                        ;DATA(I)=DATA(J)
000456                    CPXTBL  DATA,C,DJADDR,DJVAL      ;HL <- (DATA(J))
000456  041  040  005
000461  175
000462  201
000463  157
000464  022  070  001
000467  044
000470  042  026  005
000473  176
000474  043
000475  156
000476  147
000477  042  034  005
000502                    PUSH    D
000502  325
000503                    XCHG
000503  353
000504                    LHLD    DIADDR
000504  052  030  005
000507                    MOV     M,D
000507  162
000510                    INX     H
000510  043
000511                    MOV     M,E
000511  163
                        ;DATA(J)=TEMP
000512                    LHLD    TEMP
000512  052  024  005
000515                    XCHG
000515  353
000516                    LHLD    DJADDR
000516  052  026  005
000521                    MOV     M,D
000521  162
000522                    INX     H
000522  043
000523                    MOV     M,E
000523  163
000524                    POP     D
```

Program C(1) (continued)

```
000524  321
                        ;50   N1=(N3+1)/2
000525                  S50:    MVI     A,N3+1          ;A <- N3+1
000525  076  040
000527                          ANA     A               ;RESET CARRY
000527  247
000530                          RAR                     ;A <- A/2
000530  037
000531                          MOV     D,A
000531  127
                        ;60   IF(J-N1.LT.0) GO TO 80
000532                  S60:    MOV     A,C
000532  171
000533                          CMP     D
000533  272
000534                          JM      S80
000534  372  164  001
                        ;70   J=J-N1
000537                  S70:    SUB     D
000537  222
000540                          MOV     C,A
000540  117
                        ;N1=N1/2
000541                          MOV     A,D
000541  172
000542                          ANA     A               ;RESET CARRY
000542  247
000543                          RAR                     ;A <- A/2
000543  037
000544                          RAR                     ;TEST FOR ODD
000544  037
000545                          JNC     S75             ;SKIP IF EVEN
000545  322  155  001
000550                          RAL                     ;RESTORE VALUE
000550  027
000551                          INR     A               ;A <- A+1
000551  074
000552                          JMP     S76
000552  303  156  001
000555                  S75:    RAR                     ;RESTORE VALUE
000555  027
000556                  S76:    MOV     D,A
000556  127
                        ;IF(N1-1.GE.0) GO TO 60
000557                          CPI     3               ;N1 IS IN A
000557  376  003
000561                          JP      S60
000561  362  132  001
                        ;80   J=J+N1
000564                  S80:    MOV     A,C
000564  171
000565                          ADD     D
000565  202
000566                          MOV     C,A
000566  117
                        ;I=I+1
000567                          INR     B
```

Program C(1) (continued)

```
000567  004
000570                          INR     B
000570  004
                              ;IF(I-N3-1.LT.0)  GO TO 45
000571                          MOV     A,B
000571  170
000572                          DCR     A
000572  075
000573                          CPI     N3
000573  376     037
000575                          JM      S45
000575  372     022     001
                      ;
                      ;       2 POINT FOURIER TRANSFORMS & PHASE SHIFTS
                      ;
                      ;  REGISTER B HOLDS THE VARIABLE I
                      ;  REGISTER C HOLDS THE VARIABLE J
                      ;  REGISTER D HOLDS THE VARIABLE N1
                      ;  REGISTER E HOLDS THE VARIABLE N2
                      ;  REGISTER H HOLDS THE VARIABLE M1
                      ;  REGISTER L HOLDS THE VARIABLE N0
                      ;90   K=0
000600                S90:    LXI     H,K
000600  041     351     004
000603                          MVI     M,0
000603  066     000
                              ;M2=1
000605                          INX     H               ;M2 FOLLOWS K IN MEMORY
000605  043
000606                          MVI     M,2             ;2 BYTES/ COMPLEX VALUE
000606  066     002
                      ;   M1=0
000610                          MVI     H,0
000610  046     000
                      ;100  IF(M2-N3-1.GT.0) GO TO 150
000612                S100:   PUSH    H
000612  345
000613                          LXI     H,M2
000613  041     352     004
000616                          MOV     A,M             ;A <- (M2)
000616  176
000617                          DCR     A
000617  075
000620                          DCR     A
000620  075
000621                          CPI     N3
000621  376     037
000623                          POP     H
000623  341
000624                          JP      MATUDE          ;FFT FINISHED.  FIND MAGNITUDES.
000624  362     142     003
                      ;       IF(M1.EQ.0) GO TO 110
000627                          MOV     A,H
000627  174
000630                          ANA     A
000630  247
000631                          JZ      S110
```

Program C(1) (continued)

```
000631  312  243  001
                        ;105  K=K+1
000634                  S105:   PUSH    H
000634  345
000635                          LXI     H,K
000635  041  351  004
000640                          INR     M
000640  064
000641                          INR     M               ;K INDEXES TABLE (2BYTES/VALUE)
000641  064
000642                          POP     H
000642  341
                        ;110  NO=M1+1
000643                  S110:   MOV     A,H             ;H=M1, L=NO
000643  174
000644                          ADI     2
000644  306  002
000646                          MOV     L,A
000646  157
                        ;N1=M2
000647                          PUSH    H               ;SAVE M1 & NO
000647  345
000650                          LXI     H,M2
000650  041  352  004
000653                          MOV     D,M
000653  126
000654                          POP     H               ;RESTORE M1 & NO
000654  341
                        ;120  N2=2*N1
000655                  S120:   MOV     A,D
000655  172
000656                          ADD     A               ;A <- 2*A
000656  207
000657                          MOV     E,A
000657  137
                        ;IF(N1-N3-1.GE.0) GO TO 140
000660                          MOV     A,D
000660  172
000661                          DCR     A
000661  075
000662                          CPI     N3
000662  376  037
000664                          JP      S140
000664  362  106  002
                        ;       I3=NO
000667                          MOV     A,L             ;L=NO
000667  175
000670                          PUSH    H
000670  345
000671                          LXI     H,I3
000671  041  353  004
000674                          MOV     M,A             ;I3 <- NO
000674  167
000675                          POP     H
000675  341
                        ;125   I=I3
000676                  S125:   PUSH    H
```

Program C(1) (continued)

```
000676    345
000677                              LXI     H,10
000677    041   353   004
000702                              MOV     B,M
000702    106
000703                              POP     H
000703    341
                          ;130  J=1+N1
000704                    S130:     MOV     A,B
000704    170
000705                              ADD     D
000705    202
000706                              MOV     C,A
000706    117
                          ;TEMP=DATA(J)
000707                              PUSH    H               ;SAVE REGISTERS
000707    345
000710                              CPXTBL  DATA,C,DJADDR,DJVAL     ;HL <- (DATA(J))
000710    041   040   005
000713    175
000714    201
000715    157
000716    322   322   001
000721    044
000722    042   026   005
000725    176
000726    043
000727    156
000730    147
000731    042   034   005
000734                              SHLD    TEMP
000734    042   024   005
000737                              POP     H               ;RESTORE REGISTERS
000737    341
                          ;IF(M1.EQ.0) GO TO 135
000740                              MOV     A,M
000740    174
000741                              ANA     A               ;AFFECTS FLAGS
000741    247
000742                              PUSH    H
000742    345
000743                              PUSH    D
000743    325
000744                              JZ      S135
000744    312   363   001
                          ;132  TEMP=(0,1)*TEMP*TABLE(K)
000747                    S132:     PUSH    B
000747    305
000750                              LHLD    TEMP            ;HL <- (TEMP)
000750    052   024   005
000753                              SUB     A               ;A <- 0
000753    227
000754                              SUB     L               ;A <- -IMAG(TEMP)
000754    225
000755                              MOV     D,A             ;DE <- TEMP*(0,1)
000755    127
000756                              MOV     E,M
```

Program C(1) (continued)

```
000756  134
000757                          CALL    MULCON          ;TEMP=DE*TABLE(K)
000757  315     215     002
000762                          POP     B
000762  301
                        ;135  DATA(J)=DATA(I)-TEMP
000763                  S135:   LHLD    TEMP            ;HL <- TEMP
000763  052     024     005
000766                          XCHG                    ;DE <- TEMP
000766  353
000767                          CPXTBL  DATA,B,DIADDR,DIVAL     ;HL <- (DATA(I))
000767  041     040     005
000772  175
000773  200
000774  157
000775  322     001     002     . BYTE ^O322,ADR1,ADR2
001000  044
001001  042     030     005
001004  176
001005  043
001006  156
001007  147
001010  042     036     005
001013                          MOV     A,H
001013  174
001014                          SUB     D               ;A <- (REAL(DATA(A))-(TEMP))
001014  222
001015                          MOV     H,A
001015  147
001016                          MOV     A,L
001016  175
001017                          SUB     E               ;A <- IMAG(DATA(I))-(TEMP))
001017  223
001020                          MOV     L,A             ;HL <- (DATA(I))-(TEMP)
001020  157
001021                          PUSH    D
001021  325
001022                          XCHG
001022  353
001023                          LHLD    DJADDR
001023  052     026     005
001026                          MOV     M,D
001026  162
001027                          INX     H
001027  043
001030                          MOV     M,E
001030  163
001031                          POP     D
001031  321
                        ;DATA(I)=DATA(I)+TEMP
001032                          LHLD    DIVAL           ;HL <- (DATA(I))
001032  052     036     005
001035                          MOV     A,D
001035  172
001036                          ADD     H
001036  204
001037                          MOV     H,A
```

Program C(1) (continued)

```
001040                          MOV     A,E
001040  173
001041                          ADD     L
001041  205
001042                          MOV     L,A
001042  157
001043                          XCHG
001043  353
001044                          LHLD    DIADDR
001044  052     030     005
001047                          MOV     M,D
001047  162
001050                          INX     H
001050  043
001051                          MOV     M,E
001051  163
001052                          POP     D
001052  321
001053                          POP     H               ;RESTORE REGISTERS
001053  341
                        ;       IF(M1.NE.0) GO TO 160
001054                          MOV     A,H             ;A <-- (M1)
001054  174
001055                          ANA     A               ;SET FLAGS
001055  247
001056                          JNZ     S160
001056  302     142     002
                        ;I=1+N2
001061                          MOV     A,B
001061  170
001062                          ADD     C
001062  203
001063                          MOV     B,A
001063  107
                        ;IF(1-N3-1.LE.0) GO TO 130
001064                          DCR     A
001064  075
001065                          DCR     A
001065  075
001066                          CPI     N3
001066  376     037
001070                          JM      S130
001070  372     304     001
                        ;130  NO=2*NO-1
001073                  S130:   MOV     A,L
001073  175
001074                          ADD     A               ;A <-- 2*A
001074  207
001075                          SUI     2
001075  326     002
001077                          MOV     L,A
001077  157
                        ;N1=2*N1
001100                          MOV     A,D
001100  172
001101                          ADD     A
```

Program C(1) (continued)

```
001102                                  MOV     D,A
001102  127
                                ;GO TO 120
001103                                  JMP     S120
001103  303     255     001
                        ;140    M1=M1+2
001106                          S140:   MOV     A,H
001106  174
001107                                  ADI     4
001107  306     004
001111                                  MOV     H,A
001111  147
                                ;IF(2*M1-M2.LE.0) GO TO 105
001112                                  ADD     A               ;A=2*M1
001112  207
001113                                  DCR     A
001113  075
001114                                  PUSH    H
001114  345
001115                                  LXI     H,M2
001115  041     352     004
001120                                  CMP     M               ;2*M1-M2
001120  276
001121                                  POP     H
001121  341
001122                                  JM      S105
001122  372     234     001
                        ;142    M1=1
001125                          S142:   MVI     H,2
001125  046     002
                        ;145    M2=2*M2
001127                          S145:   PUSH    H
001127  345
001130                                  LXI     H,M2
001130  041     352     004
001133                                  MOV     A,M
001133  176
001134                                  ADD     A               ;A <- 2*M2
001134  207
001135                                  MOV     M,A
001135  167
001136                                  POP     H
001136  341
                                ;GO TO 100
001137                                  JMP     S100
001137  303     212     001
                        ;160    I=N1+2*(1+I3+N0)-1
001142                          S160:   PUSH    H
001142  345
001143                                  LXI     H,I3
001143  041     353     004
001146                                  MOV     A,M             ;A <- I3
001146  176
001147                                  POP     H
001147  341
001150                                  INR     A
```

Program C(1) (continued)

```
001150  074
001151                          INR     A               ;A <- I3+1
001151  074
001152                          SUB     L               ;A <- I3+1-N0
001152  225
001153                          ADD     A               ;A <- 2*(I3+1-N0)
001153  207
001154                          ADD     D               ;A <- N1+2*(I3+1-N0)
001154  202
001155                          SUB     D               ;A <- N1+2*(---)-I
001155  220
001156                          MOV     B,A
001156  107
                        ;     IF(I-I3.GT.0) GO TO 130
001157                          PUSH    H
001157  345
001160                          LXI     H,I3
001160  041   353     004
001163                          CMP     M               ;A=I,(M)=I3
001163  276
001164                          POP     H
001164  341
001165                          JZ      S165
001165  312   173     002
001170                          JP      S130
001170  362   304     001
                        ;165  I3=I3+N2
                        S165:   PUSH    H
001173
001173  345
001174                          LXI     H,I3
001174  041   353     004
001177                          MOV     A,M             ;A <- I3
001177  176
001200                          ADD     E               ;A <- I3-N2
001200  203
001201                          MOV     M,A
001201  167
001202                          POP     H
001202  341
                        ;       IF(I3-N3-1.LE.0) GO TO 125
001203                          DCR     A
001203  075
001204                          DCR     A
001204  075
001205                          CPI     N3
001205  376   037
001207                          JM      S125
001207  372   276     001
                        ;       GO TO 130
001212                          JMP     S130
001212  303   073     002
                                ;SUBROUTINE MULCON(TEMP,K)
                        ;   MULCON IS A SPECIAL ROUTINE FOR MULTIPLYING A NUMBER IN
                        ;   'TEMP' BY A CONSTANT.  THIS IS ACCOMPLISHED BY A SHIFTING PROCESS
                        ;   AND SUCCESSIVE ADDITION.  FOR EXAMPLE,  SUPPOSE CONSTANT=0.81.
                        ;   SHIFTING 'TEMP' RIGHT TWO TIMES YIELDS 0.25'TEMP'.  IF THIS SHIFTED
                        ;   VALUE IS THEN SUBTRACTED FROM THE ORIGINAL VALUE, 0.75'TEMP'
```

Program C(1) (continued)

```
                                ;   RESULTS.  NOW IF 'TEMP' IS SHIFTED RIGHT FOUR TIMES, THE
                                ;   RESULT IS 0.06*'TEMP'.  ADDING THIS TO 0.75*'TEMP' WILL YIELD 0.81*'TEMP'
                                ;   THE NUMBER OF SHIFTS REQUIRED FOR EACH CONSTANT IS STORED IN
                                ;   THE ARRAY 'TABLE'.  THE SIGN OF A VALUE IN 'TABLE' INDICATES
                                ;   WHETHER THE SHIFTED VALUE IS TO BE ADDED TO OR SUBTRACTED FROM
                                ;   THE SUB-TOTAL.
                                    ;TMPROD=TEMP
001215                          MULCON: LXI     H,K
001215  041   351   004
001220                                  MOV     C,M             ;C <- (K)
001220  116
001221                                  MVI     B,0             ;BC <- (K)
001221  006   000
001223                                  LXI     H,TABLE
001223  041   354   004
001226                                  DAD     B
001226  011
001227                                  MOV     B,D             ;DE <- TEMP FROM MAIN
001227  102
001230                                  MOV     C,E             ;BC=TMPROD
001230  113
001231                          STORE:  SHLD    TKADDR          ;TKADDR <- TABLE+(K)
001231  042   032   005
                                    ;IF(K-9.EQ.0) GO TO 210
001234                                  LXI     H,K
001234  041   351   004
001237                                  MOV     A,M             ;A <- (K)
001237  176
001240                                  CPI     22
001240  376   022
001242                                  JZ      S210
001242  312   263   002
                                    ;IF(K-5.NE.0) GO TO 220
001245                                  CPI     12
001245  376   012
001247                                  JNZ     S220
001247  302   275   002
                                    ;TMPROD=2*TEMP
001252                                  MOV     A,D             ;D <- REAL(TEMP)
001252  172
001253                                  ADD     A               ;A=2*A
001253  207
001254                                  MOV     B,A             ;B <- REAL(TMPROD)
001254  107
001255                                  MOV     A,E             ;E <- IMAG(TEMP)
001255  173
001256                                  ADD     A
001256  207
001257                                  MOV     C,A             ;C <- IMAG(TMPROD)
001257  117
                                    ;GO TO 220
001260                                  JMP     S220
001260  303   275   002
                                ;210  TMPROD=5*TEMP
001263                          S210:   MOV     A,D
001263  172
001264                                  ADD     A
```

Program C(1) (continued)

```
001264  207                     ADD     A               ;A <- 4*REAL(TEMP)
001265
001265  207                     ADD     D               ;A <- A+REAL(TEMP)
001266
001266  202                     MOV     B,A
001267
001267  107                     MOV     A,E
001270
001270  173                     ADD     A
001271
001271  207                     ADD     A
001272
001272  207                     ADD     D               ;A <- 5*IMAG(TEMP)
001273
001273  202                     MOV     C,A
001274
001274  117             ;220  NTERMS=3
001275                  S220:   MVI     H,3             ;H=NTERMS
001275  046     003
                        ;225  NTERMS=NTERMS-1
001277                  S225:   DCR     H               ;1ST LOOP=REAL, 2ND=IMAG
001277  045
                        ;IF(NTERMS.EQ.0) GO TO 250
001300                          JZ      S250
001300  312     134     003
                        ;TKVAL=TABLE(K)
001303                          PUSH    H
001303  345
001304                          LHLD    TKADDR
001304  052     032     005
001307                          MOV     A,M             ;A <- (TABLE(K))
001307  176
001310                          POP     H
001310  341
                        ;IF(TKVAL.LT.0) GO TO 230
001311                          MOV     L,A             ;L <- (TABLE(K)),L=TKVAL
001311  157
001312                          ANA     A
001312  247
001313                          JM      S230
001313  372     022     003
                        ;IF(TKVAL.EQ.0) GO TO 250
001316                          JZ      S250
001316  312     134     003
                        ;TMPROD=TMPROD+TEMP/2**TKVAL
001321                          PUSH    H
001321  345
001322                          MOV     H,L             ;PUT TKVAL IN H AND L
001322  145
001323                          MOV     A,D             ;A <- REAL(TEMP)
001323  172
001324                  LOOPD:  DCR     H               ;H IS REAL LOOP COUNTER
001324  045
001325                          JM      CTNUA           ;EXIT AFTER TKVAL LOOPS
001325  372     353     002
001330                          CPI     -1              ;TEST FOR A=-1
```

Program C(1) (continued)

```
001330  376  377                    
001332                         JNZ     LM1A
001332  302  342  002                
001335                         MVI     A,0             ;IF A=-1, SET A TO 0 AND LEAVE LOOP
001335  076  000                    
001337                         JMP     CTNUA
001337  303  353  002                
001342                 LM1A:   ANA     A               ;TEST SIGN
001342  247                    
001343                         JP      LM1
001343  362  347  002                
001346                         STC
001346  067                    
001347                 LM1:    RAR                     ;A <- A/2
001347  037                    
001350                         JMP     LOOPD
001350  303  324  002                
001353                 CTNUA:  ADD     B               ;A <- REAL[TMPROD+TEMP/2**TKVAL]
001353  200                    
001354                         MOV     B,A
001354  107                    
001355                         MOV     A,E             ;A <- IMAG(TEMP)
001355  173                    
001356                 LOOPE:  DCR     L
001356  055                    
001357                         JM      CTNUB
001357  372  005  003                
001362                         CPI     -1
001362  376  377                    
001364                         JNZ     LM2A
001364  302  374  002                
001367                         MVI     A,0
001367  076  000                    
001371                         JMP     CTNUB
001371  303  005  003                
001374                 LM2A:   ANA     A
001374  247                    
001375                         JP      LM2
001375  362  001  003                
001400                         STC
001400  067                    
001401                 LM2:    RAR
001401  037                    
001402                         JMP     LOOPE
001402  303  356  002                
001405                 CTNUB:  ADD     C               ;A <- IMAG[TMPROD+TEMP/2**TKVAL]
001405  201                    
001406                         MOV     C,A
001406  117                    
                               ;TKADDR=TKADDR+1
001407                           LHLD    TKADDR
001407  052  032  005                
001412                         INX     H
001412  043                    
001413                         SHLD    TKADDR
001413  042  032  005                
001416                         POP     H
```

Program C(1) (continued)

```
001416  341
                                ;GO TO 225
001417                            JMP     0225
001417  303     277     002
                        ;230  TKVAL=COMPLEMENT(TKVAL)
001422                  0230:     MOV     A,L
001422  175
001423                            CMA
001423  057
001424                            MOV     L,A
001424  157
                                ;TMPROD=TMPROD-TEMP/2**TKVAL
001425                            PUSH    H
001425  345
001426                            MOV     H,L             ;PUT TKVAL IN H AND L
001426  145
001427                            MOV     A,D
001427  172
001430                  LOOPDD:   DCR     H
001430  045
001431                            JM      CTNUAA
001431  372     057     003
001434                            CPI     -1
001434  376     377
001436                            JNZ     LM11A
001436  302     046     003
001441                            MVI     A,0
001441  076     000
001443                            JMP     CTNUAA
001443  303     057     003
001446                  LM11A:    ANA     A
001446  247
001447                            JP      LM11
001447  362     053     003
001452                            STC
001452  067
001453                  LM11:     RAR
001453  037
001454                            JMP     LOOPDD
001454  303     030     003
001457                  CTNUAA:   SUB     B               ;A <- REAL[TEMP/2**TKVAL-TMPROD]
001457  220
001460                            MOV     B,A
001460  107
001461                            MOV     A,E
001461  173
001462                  LOOPEE:   DCR     L
001462  055
001463                            JM      CTNUBB
001463  372     111     003
001466                            CPI     -1
001466  376     377
001470                            JNZ     LM22A
001470  302     100     003
001473                            MVI     A,0
001473  076     000
001475                            JMP     CTNUBB
```

Program C(1) (continued)

```
001475  303  111  003
001500                    LM22A:  ANA     A
001500  247
001501                            JP      LM22
001501  362  105  003
001504                            STC
001504  067
001505                    LM22:   RAR
001505  037
001506                            JMP     LOOPEE
001506  303  062  003
001511                    CTNUBB: SUB     C               ;A <- IMAG[TEMP/2**TKVAL-TMPROD]
001511  221
001512                            MOV     C,A
001512  117
001513                            SUB     A               ;A <- 0
001513  227
001514                            SUB     B               ;A <- REAL[TMPROD-TEMP/2**TKVAL]
001514  220
001515                            MOV     B,A
001515  107
001516                            SUB     A
001516  227
001517                            SUB     C               ;A <- IMAG[TMPROD-TEMP/2**TKVAL]
001517  221
001520                            MOV     C,A
001520  117
                                ;TKADDR=TKADDR+1
001521                            LHLD    TKADDR
001521  052  032  005
001524                            INX     H
001524  043
001525                            SHLD    TKADDR
001525  042  032  005
001530                            POP     H
001530  341
                                ;GO TO 225
001531                            JMP     S225
001531  303  277  002
                          ;250  TEMP=TMPROD
001534                    S250:   MOV     H,B
001534  140
001535                            MOV     L,C
001535  151
001536                            SHLD    TEMP            ;TEMP=DE*CONST(K)
001536  042  024  005
                                ;RETURN
001541                            RET
001541  311
                          ;
                          ;  FEATURE EXTRACTOR
                          ;       STORE N IN A, WHERE DATA(N/2) IS THE COMPLEX TRANSFORM
                          ;       VALUE WHOSE MAGNITUDE IS TO BE FOUND.  STORE THE MAGNITUDE
                          ;       SPECIFIED BY N IN THE ARRAY MTUDE.  HERE, ONLY THE FIRST FOUR
                          ;       MAGNITUDES WILL BE FOUND.
                          ;
001542                    MATUDE: MVI     B,2             ;POINTS TO DATA(1)
```

Program C(1) (continued)

```
001542  006  002
001544                          LXI     H,MTUDE         ;MTUDE HOLDS MAGNITUDES
001544  041  331  004
001547                          SHLD    MAGS
001547  042  325  004
001552                          CALL    MAGNTD          ;FIND MAGNITUDE OF DATA(1)
001552  315  232  003
001555                          MVI     B,4             ;POINTS TO DATA(2)
001555  006  004
001557                          LXI     H,MTUDE+1
001557  041  332  004
001562                          SHLD    MAGS
001562  042  325  004
001565                          CALL    MAGNTD          ;FIND MAGNITUDE OF DATA(2)
001565  315  232  003
001570                          MVI     B,6             ;POINTS TO DATA(3)
001570  006  006
001572                          LXI     H,MTUDE+2
001572  041  333  004
001575                          SHLD    MAGS
001575  042  325  004
001600                          CALL    MAGNTD          ;FIND MAGNITUDE OF DATA(3)
001600  315  232  003
001603                          MVI     B,8.
001603  006  010
001605                          LXI     H,MTUDE+3
001605  041  334  004
001610                          SHLD    MAGS
001610  042  325  004
001613                          CALL    MAGNTD
001613  315  232  003
001616                          MVI     B,10.
001616  006  012
001620                          LXI     H,MTUDE+4
001620  041  335  004
001623                          SHLD    MAGS
001623  042  325  004
001626                          CALL    MAGNTD
001626  315  232  003
001631                          RET                     ;RETURN TO EXECUTIVE
001631  311

                        ;
                        ;       MAGNITUDE OF FFT COMPONENT.  REGISTER A HOLDS N, SUCH THAT
                        ;       DATA(N/2) IS THE COMPLEX VALUE WHOSE MAGNITUDE IS TO
                        ;       BE STORED IN ADDRESS IN 'MAGS'.
                        ;
                        ;       MAKE BOTH VALUES POSITIVE
001632                  MAGNTD: CPXTBL  DATA,D,DIADDR,DJVAL     ;HL <- (DATA(N))
001632  041  040  005
001635  175
001636  200
001637  157
001640  322  244  003
001643  044
001644  042  030  005
001647  174
001650  043
```

Program C(1) (continued)

```
001651  156
001652  147
001653  042  034  005
001656                        MOV     A,H             ;A <- REAL(DATA(N))
001656  174
001657                        ANA     A               ;AFFECT SIGN BIT
001657  247
001660                        JP      S300
001660  362  265  003
001663                        CMA
001663  057
001664                        INR     A               ;A <- -A
001664  074
001665                S300:   MOV     C,A             ;C <- ABS(REAL(DATA(N))
001665  117
001666                        MOV     A,L             ;A <- IMAG(DATA(N))
001666  175
001667                        ANA     A               ;TEST SIGN BIT
001667  247
001670                        JP      S310
001670  362  275  003
001673                        CMA
001673  057
001674                        INR     A
001674  074
001675                S310:   MOV     L,A             ;L<- ABS(IMAG(DATA(N))
001675  157
                      ;       TEST FOR BOTH VALUES = 0.
001676                        ANA     A               ;IS ABS(IMAG)=0?
001676  247
001677                        JNZ     S315
001677  302  314  003
001702                        CMP     C               ;ABS(IMAG)=0; IS ABS(REAL)=0?
001702  271
001703                        JNZ     S315
001703  302  314  003
001706                        MVI     H,0
001706  046  000
001710                        MOV     L,H             ;BOTH VALUES 0
001710  154
001711                        JMP     S455            ;HL <- 0
001711  303  317  004
                      ;       FIND LARGER VALUE.  MOVE LARGER INTO HL, SMALLER INTO C.
001714                S315:   CMP     C               ;ABS(IMAG)-ABS(REAL)
001714  271
001715                        JP      S320            ;DO NOT SWITCH IF > OR =.
001715  362  322  003
001720                        MOV     L,C
001720  151
001721                        MOV     C,A             ;A STILL CONTAINS ABS(IMAG)
001721  117
001722                S320:   MVI     H,0             ;HL <- LARGER VALUE
001722  046  000
                      ;       MULTIPLY LARGER VALUE BY 8 (SHIFT LEFT 3 TIMES)
001724                        SHLD    LARGER          ;SAVE LARGER VALUE
001724  042  327  004
001727                        DAD     H
```

Program C(1) (continued)

```
001727  051                     DAD     H
001730                          DAD     H               ;HL <- 8*HL
001731  051                     XCHG                    ;DE <- 8*LARGER VALUE
001731  353
001732                  ;TEST FOR SMALLER VALUE = 0.  IF SO, MAGNITUDE = LARGER VALUE.
001732                  MOV     A,C             ;A <- SMALLER VALUE
001733  171
001734                  ANA     A               ;AFFECT 0 FLAG
001734  247
001735                  JNZ     DIVIDE
001735  302     346     003
001740                  LHLD    LARGER          ;HL <- LARGER VALUE
001740  052     327     004
001743                  JMP     S450
001743  303     314     004
                ;       DIVIDE 8*LARGER VALUE BY SMALLER VALUE.  DE/C=A
001746          DIVIDE: MVI     B,9.            ;B IS BIT COUNTER
001746  006     011
001750                  MOV     A,D
001750  172
001751          S330:   MOV     D,A
001751  127
001752                  MOV     A,E             ;ROTATE CARRY INTO E. ROTATE
001752  173
001753                  RAL                     ;NEXT MOST SIG. BIT TO CARRY
001753  027
001754                  MOV     E,A
001754  137
001755                  DCR     B
001755  005
001756                  JZ      S340
001756  312     373     003
001761                  MOV     A,D             ;ROTATE MOST SIG. BIT TO
001761  172
001762                  RAL                     ;HIGH ORDER QUOTIENT
001762  027
001763                  SUB     C               ;SUBTRACT DIVISOR.  IF < HIGH
001763  221
001764                  JNC     S330            ;ORDER QUOTIENT, GO TO S330
001764  322     351     003
001767                  ADD     C               ;ELSE, ADD BACK
001767  201
001770                  JMP     S330
001770  303     351     003
001773          S340:   MOV     A,E
001773  173
001774                  CMA                     ;A <- QUOTIENT
001774  057
001775                  ANA     A               ;TEST SIGN BIT
001775  247
001776                  JM      S450            ;QUOTIENT > 1 BYTE, MAG = LARGER VALUE
001776  372     314     004
002001                  PUSH    PSW
002001  365
```

Program C(1) (continued)

```
002002                                  MOV     E,C             ;SAVE SMALLER VALUE IN E
002002  131
002003                                  MOV     A,C             ;PUT SMALLER VALUE IN A
002003  171
002004                                  ANA     A               ;CLEAR CARRY FLAG
002004  247
002005                                  RAR                     ;A <-- 0.5*A
002005  037
002006                                  MOV     D,A             ;D <-- 0.5*A
002006  127
002007                                  POP     PSW             ;RESTORE QUOTIENT TO A
002007  361
                        ;       QUOTIENT IS NOW IN A AND SMALLER VALUE IS IN E, WITH 0.5*SMALLER
                        ;       VALUE IN D.  TEST A TO DETERMINE THE FACTOR THAT THE SMALLER
                        ;       VALUE WILL BE MULTIPLIED BY TO APPROXIMATE THE
                        ;       MAGNITUDE OF THE COMPLEX NUMBER.  (FACTORS ARE 1.5,2,2.5,
                        ;       3,...,9,9.5)  THIS DETERMINATION IS MADE BY COMPARING
                        ;       8*QUOTIENT WITH 8*TAN(ARCCOS 1/(C+/-0.25)).  IF 8*QUOTIENT
                        ;       > 8*TAN(ARCCOS 1/(C+0.25)), THE SMALLER VALUE IS AGAIN
                        ;       SUCCESSIVELY ADDED TO ITSELF AND 8*QUOTIENT IS COMPARED
                        ;       WITH 8*TAN(ARCCOS(1/((C+1)+0.25)).  IF 8*TAN(ARCCOS(1/(C-0.25)) <
                        ;       8*QUOTIENT < 8*TAN(ARCCOS(1/(C+0.25)) THEN 0.5*SMALLER
                        ;       VALUE IS ADDED TO THE SUCCESSIVE ADDITION SUM AND THAT RESULTING
                        ;       VALUE IS RETURNED AS THE MAGNITUDE.  IF 8*QUOTIENT < 8*TAN(ARCCOS(1/(C-0.25))
                        ;       THE SUCCESSIVE ADDITION VALUE UP TO THAT POINT IS OUTPUT
                        ;       AS THE MAGNITUDE.
                        ;
002010                                  CPI     11.             ;SHOULD MAG BE > 1.5*E?
002010  376    013
002012                                  JP      S360            ;IF SO, SKIP.
002012  362    023     004
002015                                  MOV     A,E             ;A=E
002015  173
002016                                  ADD     D               ;A=1.5E
002016  202
002017                                  MOV     C,A
002017  117
002020                                  JMP     S460            ;STORE MAGNITUDE
002020  303    320     004
002023                          S360:   MOV     B,A             ;SAVE QUOTIENT
002023  107
002024                                  MOV     A,E
002024  173
002025                                  ADD     E
002025  203
002026                                  MOV     C,A             ;C=2E
002026  117
002027                                  MOV     A,B             ;RESTORE QUOTIENT
002027  170
002030                                  CPI     20.             ;SHOULD MAG BE =3?
002030  376    024
002032                                  JP      S370
002032  362    053     004
002035                                  CPI     16.             ;SHOULD MAG BE =2.5E?
002035  376    020
002037                                  JM      S460            ;IF NOT, STORE MAGNITUDE
```

Program C(1) (continued)

```
002037  372  320  004                JZ    0460
002042                                MOV   A,C
002042  312  320  004
002045                                ADD   D
002045  171
002046                                MOV   C,A          ;C=2.5E
002046  202
002047                                JMP   0460
002047  117
002050                                MOV   A,C
002050  303  320  004          0370:  ADD   E
002053                                MOV   C,A          ;C=3E
002053  171
002054                                MOV   A,B
002054  203
002055                                CPI   29.          ;SHOULD MAG BE > 3.5?
002055  117
002056                                JP    0360
002056  170
002057                                CPI   25.          ;SHOULD MAG BE = 3.5?
002057  376  035
002061                                JM    0460
002061  362  102  004
002064                                JZ    0460
002064  376  031
002066                                MOV   A,C
002066  372  320  004
002071                                ADD   D
002071  312  320  004
002074                                MOV   C,A          ;C=4E
002074  171
002075                                MOV   A,B
002075  202
002076                                CPI   37.          ;SHOULD MAG BE > 4.5?
002076  117
002077                                JP    0390
002077  303  320  004
002102                                CPI   33.          ;SHOULD MAG BE = 4.5?
002102  171
002103                                JM    0460
002103  203
002104                                JZ    0460
002104  117
002105                                MOV   A,C
002105  170
002106                                ADD   E
002106  376  045
002110                                MOV   C,A          ;C=5E
002110  362  131  004
002113                                MOV   A,B
002113  376  041
002115                                CPI   41.          ;SHOULD MAG BE > 5.5?
002115  372  320  004
002120                                JP    0420
002120  312  320  004
002123                                CPI   35.          ;SHOULD MAG BE = 5.5?
002123  171
```

Program C(1) (continued)

```
002124  202
002125                          MOV     C,A
002125  117
002126                          JMP     S460
002126  303     320     004
002131                  S390:   MOV     A,C
002131  171
002132                          ADD     E
002132  203
002133                          MOV     C,A             ;C=5E
002133  117
002134                          MOV     A,B
002134  170
002135                          CPI     45.             ;SHOULD MAG BE > 5.5?
002135  376     055
002137                          JP      S400
002137  362     160     004
002142                          CPI     41.             ;SHOULD MAG BE = 5.5?
002142  376     051
002144                          JM      S460
002144  372     320     004
002147                          JZ      S460
002147  312     320     004
002152                          MOV     A,C
002152  171
002153                          ADD     D
002153  202
002154                          MOV     C,A
002154  117
002155                          JMP     S460
002155  303     320     004
002160                  S400:   MOV     A,C
002160  171
002161                          ADD     E
002161  203
002162                          MOV     C,A             ;C=6E
002162  117
002163                          MOV     A,B
002163  170
002164                          CPI     53.             ;SHOULD MAG BE > 6.5?
002164  376     065
002166                          JP      S410
002166  362     207     004
002171                          CPI     49.             ;SHOULD MAG BE = 6.5?
002171  376     061
002173                          JM      S460
002173  372     320     004
002176                          JZ      S460
002176  312     320     004
002201                          MOV     A,C
002201  171
002202                          ADD     D
002202  202
002203                          MOV     C,A
002203  117
002204                          JMP     S460
```

Program C(1) (continued)

```
002207                          320     004
002207                  S410:   MOV     A,C
002207  171
002210                          ADD     E
002210  203
002211                          MOV     C,A             ;C=7E
002211  117
002212                          MOV     A,B
002212  170
002213                          CPI     61.             ;SHOULD MAG BE > 7.5?
002213  376     075
002215                          JP      S430
002215  362     236     004
002220                          CPI     57.             ;SHOULD MAG BE = 7.5?
002220  376     071
002222                          JM      S460
002222  372     320     004
002225                          JZ      S460
002225  312     320     004
002230                          MOV     A,C
002230  171
002231                          ADD     D
002231  202
002232                          MOV     C,A
002232  117
002233                          JMP     S460
002233  303     320     004
002236                  S430:   MOV     A,C
002236  171
002237                          ADD     E
002237  203
002240                          MOV     C,A             ;C=8E
002240  117
002241                          MOV     A,B
002241  170
002242                          CPI     70.             ;SHOULD MAG BE > 8.5E?
002242  376     106
002244                          JP      S440
002244  362     265     004
002247                          CPI     66.             ;SHOULD MAG BE = 8.5E?
002247  376     102
002251                          JM      S460
002251  372     320     004
002254                          JZ      S460
002254  312     320     004
002257                          MOV     A,C
002257  171
002260                          ADD     D
002260  202
002261                          MOV     C,A
002261  117
002262                          JMP     S460
002262  303     320     004
002265                  S440:   MOV     A,C
002265  171
002266                          ADD     E
002266  203
```

Program C(1) (continued)

```
002267                          MOV     C,A             ;C=9E
002267  117
002270                          MOV     A,B
002270  170
002271                          CPI     76.             ;SHOULD MAG BE = LARGER VAL?
002271  376     114
002273                          JP      S450
002273  362     314     004
002276                          CPI     74.             ;SHOULD MAG BE = 9.5E?
002276  376     112
002300                          JM      S460
002300  372     320     004
002303                          JZ      S460
002303  312     320     004
002306                          MOV     A,C
002306  171
002307                          ADD     D
002307  202
002310                          MOV     C,A
002310  117
002311                          JMP     S460
002311  303     320     004
002314                  S450:   LHLD    LARGER          ;HL <- LARGER VALUE (H <- 0)
002314  052     327     004
002317                  S455:   MOV     C,L             ;C <- MAGNITUDE OF LARGER VALUE
002317  115
002320                  S460:   LHLD    MAGS
002320  052     325     004
002323                          MOV     M,C             ;ADDRESS IN MAGS
002323  161
002324                          RET
002324  311
002325                  MAGS:   DS      2
002327                  LARGER: DS      2
002331                  MTUDE:  DS      16.
002351                  K:      DS      1
002352                  M2:     DS      1
002353                  10:     DS      1
                        ;  THE NEGATIVE NUMBERS IN THIS TABLE ARE MEANT TO BE 1'S
                        ;  COMPLEMENTED IN THE MULCON SUBROUTINE.  THAT IS, THE NUMBER -2
                        ;  IN THE TABLE, WHICH IS ASSEMBLED TO 11111110 IN BINARY, WILL
                        ;  BE COMPLEMENTED IN THE PROGRAM TO YIELD 00000001.  THUS, -2 IN
                        ;  TABLE WILL REALLY BE INTERPRETED AS -1 BY THE PROGRAM.
                        ;
002354                  TABLE:  DB      0,0
002354  000
002355  000
002356                          DB      -2,0,-3,-6,2,4
002356  376
002357  000
002360  375
002361  372
002362  002
002363  004
002364                          DB      -2,5,1,4,-5,-6
002364  376
002365  005
```

Program C(1) (continued)

```
002367  004
002370  373
002371  372
002372                          DB      -2,3,-2,0,3,0
002372  376
002373  003
002374  376
002375  000
002376  003
002377  000
002400                          DB      1,2,4,0,-3,5
002400  001
002401  002
002402  004
002403  000
002404  375
002405  005
002406                          DB      -2,3,-2,4,-2,5
002406  376
002407  003
002410  376
002411  004
002412  376
002413  005
002414                          DB      -2,0
002414  376
002415  000
002416                  PREV:   DS      2
002420                  SUM:    DS      4
002424                  TEMP:   DS      2
002426                  DJADDR: DS      2
002430                  D1ADDR: DS      2
002432                  TKADDR: DS      2
002434                  DJVAL:  DS      2
002436                  D1VAL:  DS      2
002440                  DATA:   DS      34.
002502                          END
```

Program C(1) (continued)